JAVA

In 8 Hours

For Beginners
Learn Java Fast!

Ray Yao

JAVA
For Beginners

By Ray Yao

Include Tests & Answers

About the Author

Ray Yao:

Certified PHP engineer by Zend, USA

Certified JAVA programmer by Sun, USA

Certified SCWCD developer by Oracle, USA

Certified A+ professional by CompTIA, USA

Certified ASP. NET expert by Microsoft, USA

Certified MCP professional by Microsoft, USA

Certified TECHNOLOGY specialist by Microsoft, USA

Certified NETWORK+ professional by CompTIA, USA

Preface

“JAVA for Beginners” covers all essential JAVA knowledge. You can learn complete primary skills of JAVA fast and easily.
This book includes many practical Hands-On Projects. You can study JAVA coding with Hands-On Projects.

Source Code for Download

This book provides source code for download; you can download the source code for better study, or copy the source code to your favorite editor to test the programs. The download link of the source code is at the last page of this book.
Start coding today!

Table of Contents

Hour 1

Start JAVA

Install Java

In order to create any Java program, the Java development environment needs to be available on the local computer system.

JDK (also called JSDK, Java SE)

JDK (Java Development Kit) contains all Java class libraries, all tools building and running Java programs. JDK can create a Java development environment.

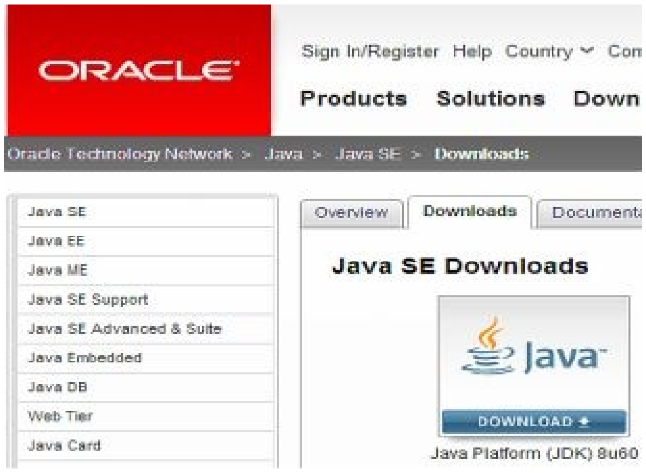
Download JDK

http://oracle.com/download

or

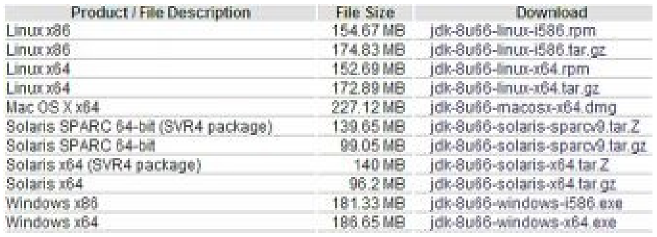
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Please click the link to download the newest version JDK at Oracle website. (Figure 1)



(Figure 1　Download JDK at Oracle Website)

If your computer is a 32-bit operating system, download the JDK with x86; If your computer is a 64-bit operating system, download the JDK with x64. (Figure 2)



Product / File Description	File Size	Download
Linux x86	154.67 MB	jdk-8u66-linux-i586.rpm
Linux x86	174.83 MB	jdk-8u66-linux-i586.tar.gz
Linux x64	152.69 MB	jdk-8u66-linux-x64.rpm
Linux x64	172.89 MB	jdk-8u66-linux-x64.tar.gz
Mac OS X x64	227.12 MB	jdk-8u66-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.65 MB	jdk-8u66-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.05 MB	jdk-8u66-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140 MB	jdk-8u66-solaris-x64.tar.Z
Solaris x64	96.2 MB	jdk-8u66-solaris-x64.tar.gz
Windows x86	181.33 MB	jdk-8u66-windows-i586.exe
Windows x64	186.65 MB	jdk-8u66-windows-x64.exe

(Figure 2)

Please install the newest version of JDK to your computer.

Congratulation! Once JDK has been installed successfully, the system is ready to start running Java programs!

IDE (Integrated Development Environment)

IDE (Integrated Development Environment) is used to write, edit and run Java. IDE contains many Java development tools, compiler and debugger. There are many excellent free IDEs for download on internet, please download one IDE by the following links:

IDE(editor)	**Website**
Eclipse	http://www.eclipse.org
NetBeans	https://netbeans.org/
DrJava	http://www.drjava.org/
Jedit	http://www.jedit.org/
BlueJ	http://www.bluej.org/
JCreator	http://www.jcreator.com/

Please install one of the editors to your computer.

(This book uses Eclipse as editor).

Congratulation! Once an IDE has been installed successfully, you can write, edit and run Java programs from now on.

What is Java?

Java is a class-based, object-oriented programming language, which is a secure, fast, and reliable program for general-purpose such as internet, database, cell phone, countless variety of equipment.

Example 1.1

```
public class Hi
{
    public static void main (String [ ] args)
    {
        System.out.print(“Hello World”);
    }
}
```

(Output: Hello World)

Explanation:

“Public class” is two keywords; they are used to declare a Java program. “Hi” is a program name.

“public static void main (String [ ] args) {**…**}” is used to declare a main method.

“main” method contains main program codes to execute.

“String [ ] args” is used to accept the input manually.

“System.out.print( )” is used to display or print.

Each Java command should be ended with semicolon (**;**).

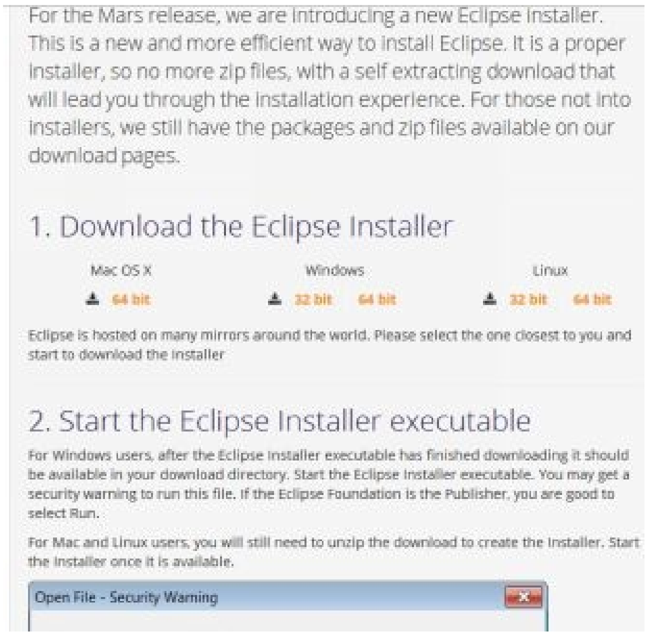
Run First Program

Eclipse is one of the most popular Java IDE (Integrated Development Environment). It is used to write, edit, debug and run Java program.

Download Eclipse Installer

http://www.eclipse.org/downloads/

Please click the link to download the newest version Eclipse Installer at Eclipse website. (Figure 1)



For the Mars release, we are introducing a new Eclipse Installer. This is a new and more efficient way to install Eclipse. It is a proper installer, so no more zip files, with a self extracting download that will lead you through the installation experience. For those not into installers, we still have the packages and zip files available on our download pages.

1. Download the Eclipse Installer

Mac OS X	Windows	Linux
64 bit	32 bit 64 bit	32 bit 64 bit

Eclipse is hosted on many mirrors around the world. Please select the one closest to you and start to download the Installer

2. Start the Eclipse Installer executable

For Windows users, after the Eclipse Installer executable has finished downloading it should be available in your download directory. Start the Eclipse Installer executable. You may get a security warning to run this file. If the Eclipse Foundation is the Publisher, you are good to select Run.

For Mac and Linux users, you will still need to unzip the download to create the Installer. Start the Installer once it is available.

Open File - Security Warning

(Figure 1 Download Eclipse installer at Eclipse Website.)

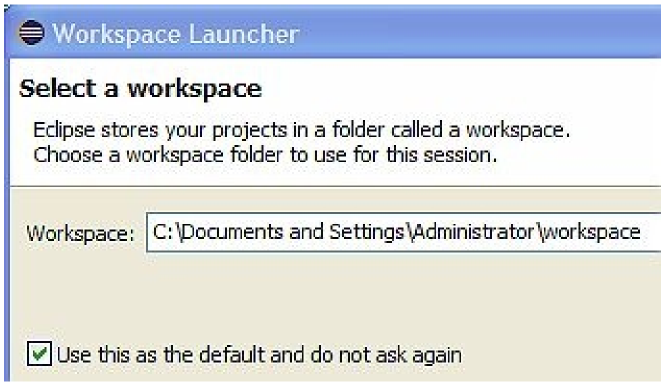
Uncompress Eclipse File

Note**:** instead of installation, the Eclipse should be uncompressed in a directory you prefer. For example, **C:\Eclipse**. Please create a shortcut of **eclipse.exe** to desktop.

(Restate**:** Eclipse doesn’t need to be installed, but it should be uncompressed in a directory.)

Run Eclipse

Double click **eclipse.exe** to run Eclipse, set the default workplace, check the box of “Use this as the default and do not ask again”, and click OK, as following Figure 2 and Figure 3.



Workspace Launcher

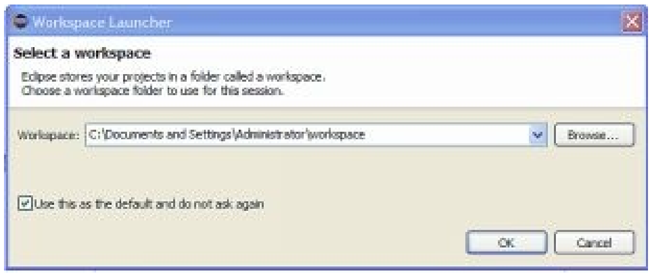
Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Documents and Settings\Administrator\workspace

☑ Use this as the default and do not ask again

(Figure 2)



Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Documents and Settings\Administrator\workspace

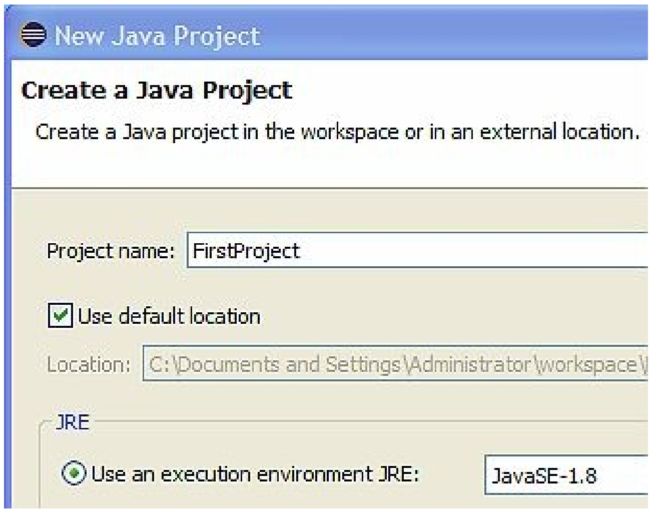
☑ Use this as the default and do not ask again

(Figure 3)

Write & Run First Java Program

Step 1: Create a Java Project

Start up Eclipse > File > New > Java Project > Project Name > “FirstProject” > Finish. (Figure 4)



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: FirstProject

[✓] Use default location

Location: C:\Documents and Settings\Administrator\workspace\

JRE

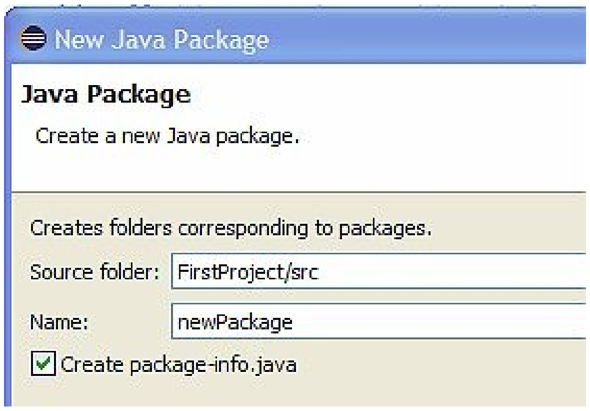
(●) Use an execution environment JRE: JavaSE-1.8

(Figure 4)

Step 2: Create a Package

Package is used to contain various categories of Java files. Different package contains different Java files.

Select project “FirstProject” > new > package > name > “newPackage” > Finish. (Figure 5)



New Java Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

Source folder: FirstProject/src

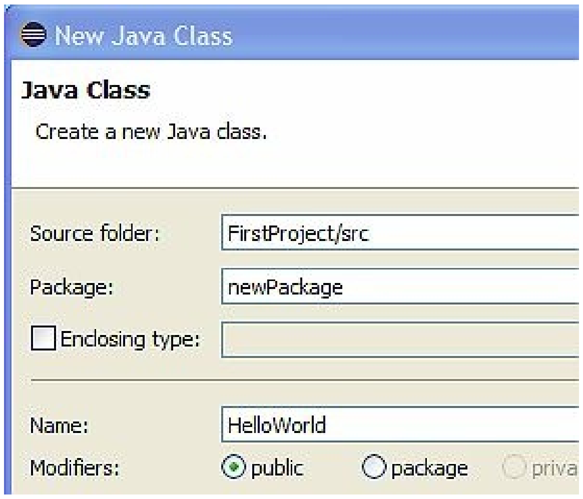
Name: newPackage

[✓] Create package-info.java

(Figure 5)

Step 3: Create a Class

Select package “newPackage” > new > class > name > “HelloWorld” > Finish. (Figure 6)



New Java Class

Java Class

Create a new Java class.

Source folder: FirstProject/src

Package: newPackage

☐ Enclosing type:

Name: HelloWorld

Modifiers: [✓ public] package priva

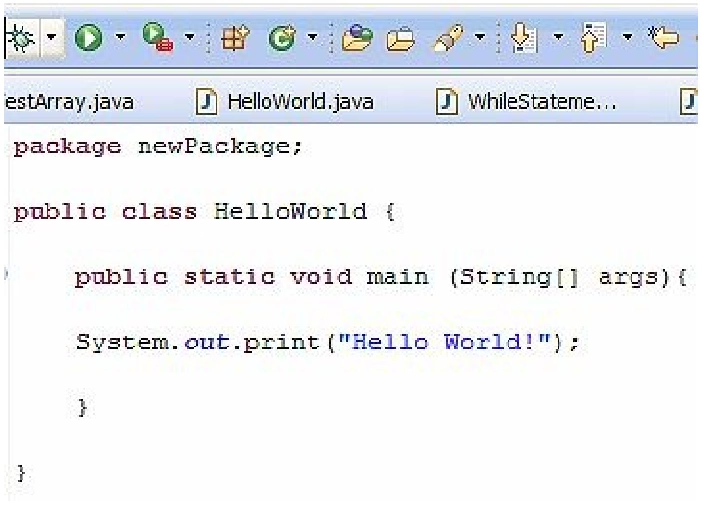
(Figure 6)

Step 4: Write Java Codes

Write Java codes in Eclipse. (Figure 7)

Example 1.2

```
package newPackage;

public class HelloWorld {
	public static void main (String[] args){

	System.out.print("Hello World!");
}
}
```



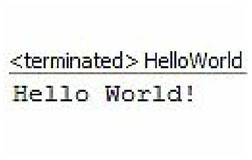
```
package newPackage;

public class HelloWorld {

    public static void main (String[] args){

    System.out.print("Hello World!");

    }

}
```

(Figure 7 Java program in Eclipse)

Step 5: Run the Program

Click White Triangle Green Button in tool bar to run the program, you can see the output: (Figure 8)

Output:



```
<terminated> HelloWorld
Hello World!
```

(Figure 8)

Congratulation! The first Java program run by **Eclipse** is successful!

Java Comments

```
//

/*  ......  */
```

Java comments is used to explain the current code, describe what this code is doing. Java compiler will ignore comments.

// is used to single line comment.

/*…*/ is used to multi line comments

Example 1.3

```
System.out.println (“Hello World”);  // show “Hello World”
/*  System.out.println( ) is a Java output command, meaning display or print.
*/
```

Explanation:

// show “Hello World” is a single line comment.

/* System.out.println( ) is a Java output command, meaning display or print. ***/** is a multi line comment.

Output Commands

```
System.out.print ( );  // print the result at the same line.

System.out.println ( );  // print the result at the different
line.
```

Example 1.4

```
System.out.print( "1 " );
System.out.print( "2 " );
System.out.print( "3" );
```

Output: 1 2 3

```
System.out.println( "1 " );
System.out.println( "2 " );
System.out.println( "3" );
```

Output:

```
1
2
3
```

Explanation:

System.out.print( ); will print the result at the same line.

System.out.println( ) will print the result at the different line.

Escaping Characters

The “ \ ” backslash character can be used to escape characters.

```
\n outputs content to the next new line.
\r makes a return
\t makes a tab
\’ outputs a single quotation mark.
\” outputs a double quotation mark.
```

Example 1.5

```
public class Hi{
    public static void main ( String [ ] args ) {  // main function
       System.out.print("  \" Hello World  \"  ");
}}       //  \"output a double quotation mark
```

Output: "Hello World"

Explanation:

\**”** outputs a double quotation mark. Note that **“**Hello World**”** has a double quotation with it. Another sample:

System.out.print (“Hello **\t\t\t** World”);   // Note it has three taps. (Output: Hello   World)

Java Keywords

Some words are only used by Java language itself. They may not be used when choosing identifier names for variable, function, and labels. The following words are part of Java reserved words:

abstract	boolean	break
byte	case	catch
char	class	const
continue	default	do
double	else	extends
false	final	finally
float	for	if
implements	import	int
interface	new	null
package	private	protected
public	return	static
String	super	switch
this	throw(s)	true
try	void	while

Example 1.6

```
const        continue      // Java keywords
```

Explanation:

Above words “const”, “continue” is Java reserved words, which may be not used as variable name, function name, and label name.

Data Types

Java data types are listed in the following:

Data Types	**Explanation**
String	a string of characters with double quotes
char	a single character with single quotes
int	an integer number
float	a floating point number
boolean	a value with true or false
double	a large or floating point number

Example 1.7

```
“Very Good”      // String type
‘S’              // char type
168              // int type
8f               // float type
true             // boolean type
```

Explanation:

The data type of “Very Good” is a String.

The data type of ‘S’ is a char.

The data type of 168 is an int

The data type of 8f is a float

The data type of true is a boolean.

Create a Variable

Variable is a symbolic name associated with a value.

Using following syntax can create a variable and value.

```
dataType variableName = value;
```

Example 1.8

```
public static void main ( String [ ] args ) {   // main function
String mystring = “Hello World”;      // mystring is a variable
char mychar = ‘m’;                          // mychar is a variable
int myinteger = 168168;          // myinteger is a variable
double myflt = 28.98;            // myflt is a variable
boolean mybool = true;         // mybool is a variable
}
```

Explanation:

mystring is a name of variable, its value is “Hello World”.

myinteger is a name of variable, its value is 168168.

mybool is a name of variable, its value is true.

“public static void main (String [ ] args) {   }” declares a main method.

Arithmetical Operators

Operators	**Running**
+	add, connect strings
-	subtract
*	multiply
/	divide
%	get modulus
++	increase 1
- -	decrease 1

Example 1.9

```
public class TryClass{
public static void main (String [ ] args){
int sum = 100 + 200;
String str = "We love" + "Java";
int incre = 10;  incre = ++ incre;
System.out.println ( sum );   // Output:  300
System.out.println ( str );   //  Output:  We love Java.
System.out.println ( incre );  // Output:  11
}}
```

Explanation:

int sum = 100 + 200 means 100 plus 200.

String str = “We love” + “Java” connects two strings.

++incre increases its value by 1.

Logical Operators

Operators	**Equivalent**
&&	and
||	or
!	not

After using logical operators, the result will be true or false.

Example 1.10

```
public class TryClass{
public static void main (String [ ] args){
boolean x=true;  boolean y=false;
boolean a, b, c;
a= x && y;   System.out.print (a);   // output: false
b=x || y;   System.out.print (b);   // output: true
c=! x;   System.out.print (c);   // output:  false
}}
```

Explanation:

true && true; returns true;	true && false; returns false;	false &&false; returns false;
true II true; returns true;	true II false; returns true;	false II false; return false;
! false; returns true;	**!** true; returns false;	

Assignment Operators

Operator	**Example**	**Explanation**
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Example 1.11

```
int x=200; int y=100;
x+=y; System.out.print(x);
x-=y; System.out.print(x);
x/=y; System.out.print(x);
x%=y; System.out.print(x);
```

Explanation:

x+=y; // x=x+y, x=200+100, output 300.

x-=y; // x=x-y, x=200-100, output 100.

x/=y; // x=x/y, x=200/100, output 2.

x%y; // x=%y, x=200%100, output 0.

Comparison Operators

Operators	**Running**
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
==	equal
!=	not equal

After using comparison operators, the result will be true or false.

Example 1.12

```
int x=200; int y=100;
boolean result1=(x>y); System.out.print(result1);
boolean result2=(x<=y); System.out.print(result2);
boolean result3=(x!=y ); System.out.print(result3);
```

Explanation:

```
x>y     // test 200>100, output true.
x<=y  // test 200<=100, output false.
x!=y    // test 200!=100, output true.
```

Conditional Operator

The syntax of conditional operator looks like this:

```
(test-expression) ? (if-true-do-this) : (if-false-do-this);
```

(test-expression) such as   a<b, x!= y, m==n. etc.

Example 1.13

```
int a=100; int b=200;
String result1 = (a<b) ? ”apple” : ”banana”;
System.out.print ( result1);
```

(Output: apple)

Example 1.14

```
int a=100; int b=200;
String result2 = (a>b) ? ”apple” : ”banana”;
System.out.print ( result2 );
```

(Output: banana)

Explanation:

If the test expression returns true, then outputs “apple”.

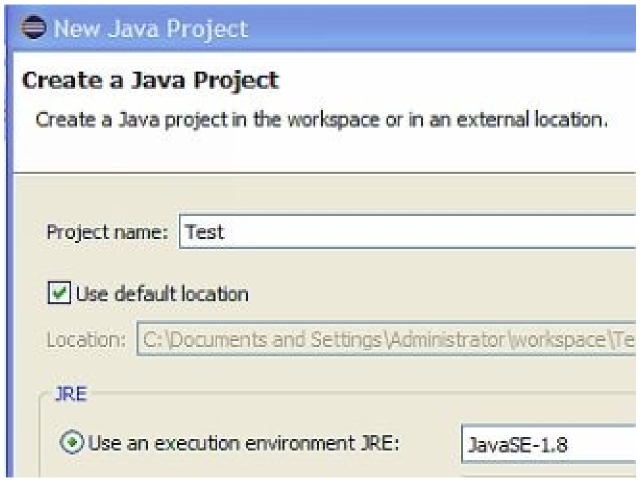
If the test expression returns false, then outputs “banana”.

Hands-On Project: Calculation

Write & Run Java Program by Eclipse

Step 1: Create a Java Project

Start up Eclipse > File > New > Java Project > Project Name > "Test" > Finish. (Figure 1)



Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: Test

☑ Use default location

Location: C:\Documents and Settings\Administrator\workspace\Te

JRE

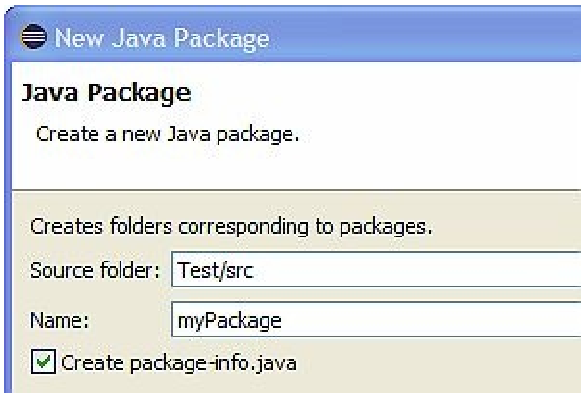
● Use an execution environment JRE: JavaSE-1.8

(Figure 1)

Step 2: Create a Package

Package is used to contain various categories of Java files. Different package contains different Java files.

Select project "Test" > new > package > name > "myPackage" > Finish. (Figure 2)



Java Package

Create a new Java package.

Creates folders corresponding to packages.

Source folder: Test/src

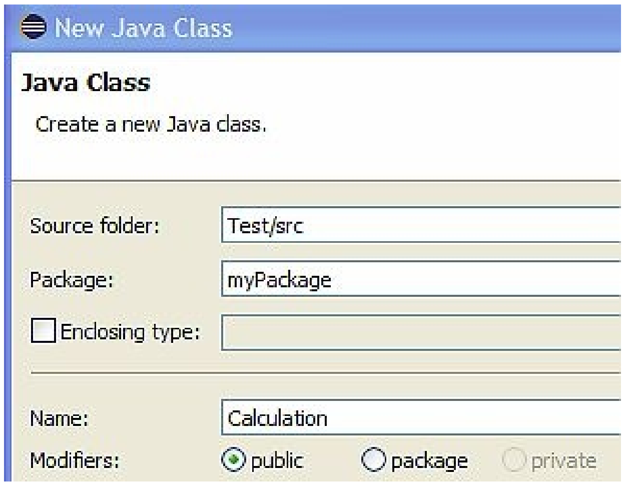
Name: myPackage

☑ Create package-info.java

(Figure 2)

Step 3: Create a Class

Select package “myPackage” > new > class > name > “Calculation” > Finish. (Figure 3)



Source folder: Test/src
Package: myPackage
Enclosing type:
Name: Calculation
Modifiers: [✓ public] package private

(Figure 3)

Step 4: Write Java Codes

Write Java codes in Eclipse.

```
package myPackage;
public class Calculation{
public static void main (String args[ ]){
int a = 100, b = 5;   // define two variables
int c = a/b*10;     // calculation
System.out.print("C = " + c);    // output the value of “c”
}
}
```

Step 5: Run the Program

Click White Triangle Green Button in tool bar to run the program, you can see the output:

Output:

```
C = 200
```

Explanation: “int c = a/b*10;” calculates the expression and assigns the result value to variable “c”.

Hour 2

Statements

If Statement

```
if ( test-expression ) {   // if true do this;   }
```

“if statement” executes codes inside { … } only if a specified condition is true, does not execute any codes inside {…} if the condition is false.

Example 2.1

```
public class TryClass{
public static void main (String [ ] args){
int a=200;
int b=100;
if (a>b) {   // if true, run the next command
System.out.print ( "a is greater than b.");
}}}
(Output:   a is greater than b.)
```

Explanation:

(a>b) is a test expression, namely (200>100), if returns true, it will execute the codes inside the { }, if returns false, it will not execute the codes inside the { }.

If-else Statement

```
if ( test-expression) {   // if true do this;   }
else  {  // if false do this;   }
```

“if...else statement” runs some code if a condition is true and another code if the condition is false

Example 2.2

```
public class TryClass{
public static void main (String [ ] args){
int a=100; int b=200;
if (a>b){    // if true do this
System.out.print ( "a is greater than b.");
}
else {    // if false do this
System.out.print ( "a is less than b");
}}}
(Output:  a is less than b.)
```

Explanation:

(a>b) is a test expression, namely (100>200), if returns true, it will output ”a is greater than b.” if returns false, it will output “a is less than b”.

Switch Statement

```
switch ( variable )
{ case 1: if equals this case, do this;  break;
  case 2: if equals this case, do this;  break;
  case 3: if equals this case, do this;  break;
  default : if not equals any case, run default code;
break;
}
```

The value of variable will compare each case first, if equals one of the “case” value; it will execute that “case” code. “break;” terminates the code running.

Example 2.3

```
public class TryClass{
public static void main (String [ ] args){
int number=20;
switch ( number ) {   // “number” value  compares each case
case 10 : System.out.print ("Running case 10");  break;
case 20 : System.out.print ("Running case 20");  break;
case 30 : System.out.print ("Running case 30");  break;
default :  System.out.print ("Running default code");  break;
}}}   (Output:  Running case 20)
```

Explanation:

The “number” value is 20; it will match case 20, so it will run the code in case 20.

For Loop

```
for( init, test-expression, increment) { // some code; }
```

“for loop” runs a block of code by specified number of times.

Example 2.4

```
public class TryClass{
public static void main (String [ ] args){
int x;
for (x = 0; x <= 5; x++) {     // repeat at most 5 times
System.out.print( x );
}}}
(Output: 012345 )
```

Explanation:

x = 0 is initializer,

x <= 5 is test-expression, the code will run at most 5 times.

x++ means that x will increase 1each loop.

After 5 times loop, the code will output 012345.

While Loop

```
while ( test-expression ) {  // some java code in here;
}
```

“while loop” loops through a block of code if the specified condition is true.

Example 2.5

```
public class TryClass{
public static void main (String [ ] args){
int counter=0;
while (counter < 8){       // run 8 tmes
System.out.print  ("&");
counter++;
}}}
( Output:  &&&&&&&& )
```

Explanation:

“counter< 8” is a test expression, if the condition is true, the code will loop less than 8 times, until the counter is 8, then the condition is false, the code will stop running.

Do-While Loop

```
do{ // some java code in here } while ( test-
expression);
```

“do...while” loops through a block of code once, and then repeats the loop if the specified condition is true.

Example 2.6

```
public class TryClass{
public static void main (String [ ] args){
int counter=0;
do {
System.out.print ( "@");
counter++;
} while (counter<8);     // run 8 times
}}
( Output: @@@@@@@@ )
```

Explanation:

“counter< 8” is a test expression, if the condition is true, the code will loop less than 8 times, until the counter is 8, then the condition is false, the code will stop running.

Break Statement

```
Break;
```

“break” keyword is used to stop the running of a loop according to the condition.

Example 2.7

```
public class TryClass{
public static void main (String [ ] args){
int num=0;
while (num<10){
if (num==5)  break;    // leave the while loop
num++;
}
System.out.print( num );
}}
( Output:  5)
```

Explanation:

“if (num==5) break;” is a break statement. If num is 5, the program will run the “break” command, the break statement will leave the while loop, then run “System.out.print(num)”.

Continue Statement

```
continue;
```

“continue” keyword is used to stop the current iteration, ignoring the following codes, and then continue the next loop.

Example 2.8

```
public class TryClass{
public static void main (String [ ] args){
int num=0;
while (num<10){
num++;
if (num==5)  continue;    // go to next while loop
System.out.print( num );
}}}
( Output: 1234678910)
```

Explanation:

Note that the output has no 5.

“if (num==5) continue;” includes “continue” command. When the num is 5, the program will run “continue”, skipping the next command “System.out.print(num)”, and then continue the next while loop.

Boolean-Expression

```
If ( bool-expression ) { statements }
while ( boolean-expression ) { statement }
```

The boolean-expression should be a test expression. For example: a>b; x<=y; m!=n; It cannot be a variable only.

Example 2.9

```
public class TryClass{
public static void main (String [ ] args){
int num=10;
if( num ){  //error! because “num” is not a bool-expression
System.out.print ("No good");
}
while ( num ){  //error! for “num” is not a bool-expression
System.out.print ("No good");
}}}
```

Explanation:

if(**num**) and while (**num**) is not correct codes, because it should be a boolean-expression inside the (   ), for example, if (num ==10) and while(num<10) are correct!

Hands-On Project: Run 100 Times

While Statement Program

Start up Eclipse > Select “myPackage” > new > class > name > WhileStatement > Finish.

Write following codes in Eclipse.

```
package myPackage;
public class WhileStatement {
        public static void main(String[ ] args) {
        int n = 1;
        int sum = 0;
        while(n <= 100){   // run 100 times
        sum += n;    // sum = sum + n
        n++;
        }
        System.out.println("Sum = " + sum);
        }
}
```

Click White Triangle Green Button in tool bar to run the program and see the output:

Output:

```
Sum = 5050
```

Explanation:

“n<=100” is a test expression, if the condition is true, the code will loop less than 100 times, until the counter is 100, then the condition is false, the code will stop running.

“sum += n;” is the same as “sum = sum + n”.

“n++;” means “n” increases 1in each loops.

Hour 3

Array & Math

Create an Array (1)

An array is a variable, which can contain multiple values.

The first method to create an array as following:

```
int array-name[ ] = {“value0”, “value1”, “value2”,
…};
```

Example 3.1

```
int myarray[ ] = { 10,20,30,40 };      // create an array
```

Explanation:

The above code creates an array named “myarray”, which has four elements:

The 1st element is myarray[0] with value 10. Key is 0.

The 2nd element is myarray[1] with value 20. Key is 1.

The 3th element is myarray[2] with value 30. Key is 2.

The 4th element is myarray[3] with value 40. Key is 3.

In array, Key’s alias is “index”. Index’s alias is “key”.

Note that index begins from 0.

Create an Array (2)

An array is a variable, which can contain multiple values.

The second method to create an array as following:

```
int array-name[ ] = new int [ number of elements ];
int array-name[index0] = “value0”;
int array-name[index1] = “value1”;
int array-name[index2] = “value2”;
```

Example 3.2

```
int myArray[ ] = new int [3];   // create an array
myArray [0] = 100;
myArray [1] = 200;
myArray [2] = 300;
```

Explanation:

Above codes create an array, the array name is “myArray”, and it has three elements: myArray [0], myArray [1], myArray [2]. Their indexes are 0, 1, and 2. Their values are 100, 200, and 300. Note that index number begins from 0.

Array Length

```
array-name.length;
```

array-name.length can return the total number of elements of an array.

Example 3.3

```
public class TryClass{
public static void main (String [ ] args){
int arr[ ] = {10,20,30};   // create an array “arr”
int num = arr.length;   // get length of the array
System.out.print ( num );
}}
( Output:  3 )
```

Explanation:

“arr.length;” is used to get the length of the array “arr”, which means the total number of elements.

Note that string.length( ) is used to get the length of a string characters, instead of array elements.

Element Value

```
int value = array[index];

//  Get a value from an element.
```

```
int array[index] = value;

//  Set a value to an element.
```

Example 3.4

```
int arr[ ] = {10, 20, 30, 40, 50};
int value = arr[2];   // Get a value from an element.
System.out.println ( value );
( Output: 30)
```

Example 3.5

```
int num[ ] = new int [5];
num[2] = 800;     // Set a value to an element.
System.out.println ( num[2]);
( Output:  800)
```

Explanation:

“int value = arr[2];” gets a value 30 from “arr[2]”, and assign to the “value” variable.

“num[2] = 800;” sets a value 800 to the element “num[2]” .

Sort Array Elements

```
Arrays.sort (array-name);
```

“Arrays.sort (array-name);” can sort the array element, and display the result in order.

Before using Arrays.sort( ), you must “import java.util.Arrays”;

Example 3.6

```
package tryPackage;
import java.util.Arrays;
public class TryClass{
public static void main (String [ ] args){
String myarray[ ] = { "Cat", "Apple", "Dog", "Boy"};
Arrays.sort (myarray);   // sort the array
for(int num=0; num<myarray.length; num++){   //run 4 times
System.out.print (myarray[num]+" ");    //num is a key
}}}
```

(Output: Apple Boy Cat Dog)

Explanation:

“Arrays.sort (myarray);” sorts the array ”myarray” by its element.
“myarray.length” returns “myarray”s length that is 4.
“for loop” runs output command four times, and display the sorted result: Apple Boy Cat Dog.

Math Methods

Method	**Returns**
abs( )	a number’s absolute value
ceil( )	an integer greater than or equal its argument
cos( )	an angle’s trigonometric cosine
exp( )	a number’s Math.E to the power
floor( )	an integer less than or equal its argument.
log( )	a number’s natural logarithm
random( )	a random positive value from 0 to 1
max( )	a greater between two numbers
min( )	a smaller between two numbers
pow( )	the first argument to the power of the second
round( )	the closest long or int
sin( )	an angle’s trigonometric sine
sqrt( )	a number’s square root
tan( )	an angle’s trigonometric tangent

Example 3.7

```
float num = 10.2f;
System.out.print( Math.round(num) );   //return an integer
(Output:  10)
```

Explanation:

Math.round(num) returns an integer of 10.2f. Result is 10.

Ceil ( ) & Floor ( )

```
Math.ceil( );
Math.floor( );
```

“Math.ceil( );” returns a closest integer that is greater than or equal to its argument.
“Math.floor( );” returns a closest integer that is less than or equal to its argument.

Example 3.8

```
public class TryClass{
public static void main (String [ ] args){
float num = 9.5f;
System.out.println( "Ceiling number is "+Math.ceil( num ) );
System.out.println( "Flooring number is "+Math.floor( num ) );
}}
( Output:  Ceiling number is 10.0
           Flooring number is 9.0)
```

Explanation:

“Math.ceil(num));” returns a closest integer that is greater than or equal 9.5f, the result is 10.0.

“Math.floor(num);” returns a closest integer that is less than or equal 9.5f, the result is 9.0.

Max ( ) & Min ( )

```
Math.max( );
Math.min( );
```

“Math.max( )” returns the greater one between two numbers.

“Math.min( )” returns the less number between two numbers.

Example 3.9

```
public class TryClass{
public static void main (String [ ] args){
int x = 100;
int y = 200;
System.out.println("Greater number is "+Math.max(x, y) );
System.out.println("Less number is "+Math.min(x, y) );
}}
```

```
 ( Output:  Greater number is 200
            Less number is 100)
```

Explanation:

“Math.max(x, y)” returns the greater number between 100 and 200, the result is 200.

“Math.min(x, y)” returns the less number between 100 and 200, the result is 100.

pow ( ) & sqrt ( )

```
Math.pow( );

Math.sqrt( );
```

“Math.pow ( );” returns the first argument raised to the power of the second argument.

“Math.sqrt ( );” returns the square root of the argument.

Example 3.10

```
public class TryClass{
public static void main (String [ ] args){
int num = 4;
System.out.println("Square number is "+Math.pow(num,2) );
System.out.println("Square root is "+Math.sqrt(num) );
}}
( Output: Squared number is 16.0
                Square root is 2.0)
```

Explanation:

“Maht.pow(num,2)” returns the first argument “4” raised to the power of the second argument “2”, the result is 16.0

“Math.sqrt(num)” returns the square root of the argument “4”, the result is 2.0

Math.PI & random( )

```
Math.PI

Math.random( )
```

Math.PI is a constant that stores the value of pi.

Math.random( ) generates a number between 0.0 to 1.0.

Example 3.11

```
public class TryClass{
public static void main (String [ ] args){
double pi = Math.PI;
double ran = Math.random( );
System.out.println("The PI value is" + pi);
System.out.println("The random number is" + ran);
}}
( Output:  The PI value is 3.141592653589793
           The random number is 0.16869328)
```

Explanation:

The value of Math.PI is 3.141592653589793.

Math.random( ) generates a random number between 0.0 to 1.0, in here the result is 0.16869328.

Hands-On Project: Start with Zero

Assign values to an array by loop statement.

Start up Eclipse > Select “myPackage” > new > class > name > TestArray > Finish.

Write following codes in Eclipse.

```
package myPackage;
public class TestArray {
public static void main(String[] args) {
int[]myArray;
myArray = new int[10];   //create an array “myArray”
for (int n = 0; n < myArray.length; n++){  // run 10 times
myArray[n] = n;    // set the value of “n” to each array element
System.out.print(myArray[n] + "");           //output each element
}
}
}
```

Click White Triangle Green Button in tool bar to run the program and see the output:

Output:

```
0 1 2 3 4 5 6 7 8 9
```

Explanation:

“myArray = new int[10];” creates an array “myArray” with 10 elements.

“for (int n = 0; n < myArray.length; n++)” is a loop statement.

“int n = 0” initializes the variable “n” as 0.

“n < myArray.length” sets the loop’s number of times.

The loop’s number of times depends on the length of myArray.

“n++” means the variable “n” increases 1 in each loops.

“myArray[n] = n;” assigns value of “n” to myArray.

Hour 4

String Processing

String Length

String is consisted of one or more characters within double quotes. String.length( ) can get the length of a string.

```
String str1 = “This is a string”;  String str2 = “null”;

int length = String.length( );
```

“This is a string” is a string. “null” is a string.

“int length = String.length( );” returns the length of a String.

Example 4.1

```
public class TryClass{
public static void main (String [ ] args){
String str = "This is a string";
int length = str.length( );    // get the string length
System.out.println( "The size of string is " + length);
}}
( Output:  The size of string is 16 )
```

Explanation:

“str.length( )” returns a length of a string.

Note that array.length returns a length of an array, without brackets.

String Connection

“+” can join two strings.

concat( ) can connect two strings.

```
str = str1 + str2;

str = str1.concat(str2);
```

Example 4.2

```
String str1 = “Very”;  String str2 = “Good”;
System.out.print( str1 + str2);   //connect two strings
```

(Output: Very Good)

```
String str1 = “Very”;  String str2 = “Good”;
System.out.print( str1.concat (str2));   //connect two strings
```

(Output: Very Good)

Explanation:

“str1 + str2” or “str1.concat (str2));” can join the str1 “Very” and str2 “Good”, and become a new string “Very Good”.

String Comparing

“str1 .equals (str2);” can compare two string values, returns true or false.

```
str1 .equals ( str2 );
```

Example 4.3

```
public class TryClass{
public static void main (String [ ] args){
String str1="Good", str2="Good";
System.out.print(str1.equals (str2));
System.out.print("Good".equals ("Good"));
}}
( Output:  true  true )
```

Explanation:

“(str1.equals (str2);” or “(“Good”.equals (”Good”);” compares two strings, if the two string’s value is the same, it returns true.

Extract a Character

“charAt (int index)” returns a character located at the String's specified index. The string indexes start from zero.

```
char result= charAt (int index);
```

Example 4.4

```
public class TryClass{
public static void main(String args[]) {
String str = "Very Good";
char result = str.charAt(5);   //search the 5th character
System.out.println( result );
}}
```

(Output: G)

Explanation:

“str.charAt(5);” search the 5th character in string “Very Good”. Note that the string index starts from zero.

The 5th character is “G”.

Locate a Character

“indexOf();” returns the index within this string of the first occurrence of the specified substring. If it does not occur as a substring, -1 is returned.

```
int index = string.indexOf ( substring );
```

Example 4.5

```
public class TryClass{
public static void main(String args[ ]) {
String str = “Very Good”;
System.out.println(str.indexOf( “y” ));  //get index of “y”
}}
( Output:  3 )
```

Explanation:

“str.indexOf(“y”)” searches the index of “y” in “Very Good”. The string index starts from zero. Therefore, it returns 3.

Extract a Substring

“string.substring( )” returns a substring from beginning index to ending index-1in string.

```
String str = string.substring(int beginIndex, int
endIndex-1);
```

Example 4.6

```
public class TryClass{
public static void main(String args[ ]) {
String Str = "Very Good";
System.out.println(Str.substring(5, 7) );  /* extract some characters from index
“5” to index “7-1” */
}}
(  Output: Go )
```

Explanation:

“Str.substring(5, 7)” returns a substring from index “5” to index “7-1” in “Very Good”. The result is “Go”.

Case Change

“String.toUpperCase( )” returns a uppercase of a string.

“String.toLowerCase( )” returns a lowercase of a string.

```
String.toUpperCase( );

String.toLowerCase( );
```

Example 4.7

```
public class TryClass{
public static void main(String args[ ]){
String Str = "Very Good"; System.out.println(Str.toUpperCase( ) );
System.out.println(Str.toLowerCase( ) );
}}
( Output:   VERY GOOD  )
( Output:   very good  )
```

Explanation:

“Str.toUpperCase( )” changes “Very Good” to “VERY GOOD”.

“Str.toLowerCase( )” changes “Very Good” to “very good”.

Character Replace

“str.replace( )” can change old characters to new characters in a string, and returns a new string

```
str.replace(oldCharacter, newCharacter);
```

Example 4.8

```
public class TryClass{
public static void main(String args[ ]){
String Str = "Very Good";
System.out.println( Str.replace("Good", "Well") );
}}      // replace “Good” with “Well”
(  Output: Very Well )
```

Explanation:

“Str.replace(“Good”, “Well”)” change the characters “Good” to characters “Well” in string “Very Good”. and return new string “Very Well”.

String Type Change

“Integer.toString(number)” can change a number to a string.

“String.trim( )” can remove space at the beginning or the end of a string.

```
Integer.toString( number);
String.trim( );
```

Example 4.9

```
public class TryClass{
public static void main(String args[ ]){
int num = 169;
String str = "      Good       ";   // note that spaces.
String str1 = Integer.toString( num );    // convert to string
String str2 = str.trim( );      // remove spaces
System.out.println( str1+" is a String" );
System.out.println( str2+"is trimmed" );
}}
( Output:  169 is a String
           Good is trimmed)
```

Explanation:

“Integer.toString(num)” converts an int number 169 to a string 169.
“str.trim( )” trims “   Good   ” to a string “Good”.

StringBuffer

StringBuffer represents a string that can be dynamically changed. If a string value may change in program running, please use StringBuffer, instead of String.

```
StringBuffer str = new StringBuffer( );
str.insert ( );
str.append ( );
str.reverse ( );
```

Example 4.10

```
StringBuffer sb = new StringBuffer ( “abcde” );
sb.insert ( 2, “123”);    // Output:  abc123de
sb.append ( “456”);    // Output:  abc123de456
sb.reverse ( );    // Output:  654ed321cba
```

Explanation:

“StringBuffer sb = new StringBuffer (“abcde”)” creates an object “sb”, its value is “abcde”.

“sb.insert (2, “123”)” inserts “123” at the index of “2”.

“sb.append (“456”)” appends “456” in the end of the string.

“str.reverse ( )” reverses the order of the string.

Hands-On Project: True or False

Comparing String Values

Start up Eclipse > Select “myPackage” > new > class > name > TwoStrings > Finish.

Write following codes in Eclipse.

```
package myPackage;

public class TwoStrings {

public static void main(String[] args) {

String s1, s2, s3, s4;


s1 = "C++ in 8 Hours";

s2 = "C++ in 8 Hours";

s3 = "Python in 8 Hours";

s4 = s2;    // assign the string value

System.out.println(s1.equals(s2));

System.out.println(s1.equals(s3));

System.out.println(s3.equals(s4));

}

}
```

Click White Triangle Green Button in tool bar to run the program and see the output:

Output:

```
true
```

false
false

Explanation:

“String s1, s2, s3, s4;” declares four string variables.

“s1.equals(s2)” compares two string’s value, if the two string’s value is the same, it returns true.

Hour 5

Method, Class & Object

Method

Method is code block that can be repeated to use.

```
type method-name( ){…..};   // declares a method.

method-name( );   // calls a method.
```

Example 5.1

```
public class TryClass{
public static void main(String args[ ]){
myMethod( ); }    // call “myMethod”
static void myMethod ( ) {   // declare a method
System.out.print( "This is a method" );
}}
```

(Output: This is a method)

Explanation:

“myMethod( );” calls a method, and outputs “This is a method”.
“public static void myMethod ( )” declares a method name “myMethod”.
“public” means it can be accessed by any class.
“static” means it is a class method.
“void” is a return type, meaning without return value.

Method & Arguments

A method sometimes has argument.

```
type method-name ( argument ) {……};
method-name ( arg );
```

Example 5.2

```
public class TryClass{
public static void main ( String [ ] args ) {
myMethod( "ABC" );   //call myMethod & pass argument
}
static void myMethod (String argument) { //declare method
System.out.print( "This is a method " + argument );
}}
( Output:  This is a method ABC )
```

Explanation:

“public static void myMethod (String argument) {…}” declares a method with argument.

“myMethod(“ABC”);” calls method “myMehod(String argument) {…}”, and pass the argument “ABC” to it. After receiving the argument “ABC”, it outputs “This is method ABC”.

Return Value

“return value” can return a value to the caller.

```
type method-name ( argument ) { return value };

method-name ( arg );
```

Example 5.3

```
public class TryClass{
public  static void main ( String [ ] args ) {
int num=10;
System.out.print ("The number is  "+myMethod( num ));
}    // "myMethod(num);" is a caller
static int myMethod ( int arg ) {
return ++arg;   // send back the value to caller
} }
```

(Output: The number is 11.)

Explanation:

“myMethod(num)” calls a method.
“int myMethod (int arg) {…}” declares a method.
“int” is a return type, it means the method returns int type.
“return ++arg ” can return the value “11” to the caller.
The output is “The number is 11”.

Class Definition

A class is a template for object, and creates an object.

```
class Class-name {
type variable;
type function-name( ) { }…
}
```

“class Class-name” defines a class.

“type variable” defines a variable in class. Variable is also known as “property”.

“type function-name( ) { }” defines a function in class. Function is also known as “method”.

Example 5.4

```
public class Color {    // declare a class Color
String c1, c2;    // declare two variable members
void brightness ( ) {  System.out.print (“……”); }  //a method
}
```

Explanation:

Above codes defines a class named “Color”, two variables named “c1” and “c2”, a method named “brightness”.

Object Declaration

Object is an instance of a class.

```
obj = new Class-name(  );
obj.variable;
obj.function-name( );
```

“obj = new Class-name( );” creates a new object named “obj” for the class.

“obj.variable;” means that “obj” references a variable.

“obj. function-name ( );” means that “obj” references a method.

Example 5.5

```
Color obj = new Color(  );   // create an object "obj"
obj.c1= "yellow";   // an object references a variable
obj.c2="purple";     // an object references a variable
obj.brightness ( );   // an object references a method
```

Explanation:

“Color obj = new Color;” creates an object named” obj”, then references variable “c1” and “c2”.

“obj.brightness ( );” calls the function “brightness ( );”(in previous page), and returns “blue”.

Class & Object

Example:

```
public class Color {     // define a class
String c1, c2;
void brightness ( ) {
System.out.println("This flower is " + c1);
System.out.println("That flower is " + c2);
}
public static void main(String[] args) {
Color obj = new Color(  );     // create an object
obj.c1= "yellow.";   obj.c2="purple.";     // references
obj.brightness ( );      // an object references a method
}}
```

Output:

```
This flower is yellow.
That flower is purple.
```

Explanation:

“public class Color { }” defines a class.

“Color obj = new Color( );” creates an object.

Constructor

Constructor is used to initialize variable. Constructor name is the same as class name.

```
class Class-name{

Class-name( ) { …}      // constructor for initialization

}
```

class Class-name{ …} defines a class.

Class-name( ) { …} declares a constructor.

Example 5.6

```
public class Color {  // create a class
String c1, c2;
Color ( ) {  c1=”yellow”;  c2=”purple”; }  // constructor
}
```

Explanation:

“Color ( ) { c1=”yellow”; c2=”purple”; }” is a constructor. It initializes variable c1 as “yellow” and c2 as “purple”. when an object is created the constructor will be called automatically.

Constructor Example

Example:

```
public class Color {   // define a class
String c1, c2;
Color ( ) {c1="yellow";  c2="purple";}   // constructor
void brightness ( ) {
System.out.println("This flower is " + c1);
System.out.println("That flower is " + c2);
}
public static void main(String[] args) {
Color obj = new Color(  );  //create an object, call constructor
obj.brightness ( );
}}
```

Output:

```
This flower is yellow.
That flower is purple.
```

Explanation:

“Color ( ) {c1="yellow"; c2="purple";}” creates a constructor.

“Color obj = new Color( );” creates an object, and calls the constructor automatically.

Overloading

Overloading means that there are two or more same-name methods in a class, and their arguments are different.

Example 5.7

```
package correctPackage;

class Color{  String x,y;   // declare a class
Color ( String a, String b) { x=a; y=b; }   //overloading
Color ( ) {  x="yellow ";  y="purple ";}    //overloading

public static void main (String args[ ]) {
Color obj1=new Color("green ", "orange ");
Color obj2=new Color(  );
System.out.println(obj1.x +  obj1.y);
System.out.println(obj2.x +  obj2.y);
}
} (  Output:  green  orange
              yellow  purple )
```

Explanation:

In this class, there are two methods named “Color( )” with different argument, which are known as “overloading”.

“this” Keyword (1)

“this” is used to represent a current object.

Example 5.8

```
package correctPackage;
class MyClass {
int num;
void test(int num) {
 this.num=num;     // “this” represents the “obj”
System.out.println( this.num );
};
public static void main(String[ ] args) {
MyClass obj = new MyClass( );
obj.test(10);
}
}
```

(Output: 10)

Explanation:

In above, when “obj.test(10)” calls “test( ) { }” method, “this” signifies the object “obj”.

“this” Keyword (2)

“this” can present another same-name constructor.

Example 5.9

```
class MyClass {
int x, y;
MyClass ( ) {
this ( 100, 200 );  //"this" presents MyClass ( int a, int b){ }
}
MyClass ( int a, int b){
x=a;  y=b;
}
```

Explanation:

“**this** (100, 200)” represents another same-name constructor “MyClass (int a, int b){ }”.

Note that “this” cannot represent non-constructor method.

Instance & Local variable

Instance variable is defined in a class. It works in the current class.

Local variable is defined in a method. It works in the current method.

Example 5.10

```
class Point {
int x= 100, y = 200;   //  instance variable
void test( ){
int x = 1, y = 2;     //  local variable
}
}
```

Explanation:

“int **x**= 100, **y** = 200;” defines two instance variables inside a class.

“int **x** = 1, **y** = 2;” defines two local variables inside a method.

Hands-On Project: Contact Info.

Class and Object

Start up Eclipse > Select “myPackage” > new > class > name > Message > Finish. Write following codes in Eclipse.

```
package myPackage;
public class Message {    // declare a class
String name, email, phone;
public Message(String theName, String theEmail, String thePhone){    // declare a constructor for initialization
name = theName;
email = theEmail;
phone = thePhone;
}
void displayEmail(){      // declare a function
System.out.println("Email:" + email);
}
void displayPhone(){      // declare a function
System.out.println("Phone:" + phone);
}
public static void main(String[] args) {    // main function
Message JACK = new Message ("JACK", "JACK@xxx.xxx", "678-xxx-9999");   //create an object JACK, and call constructor
System.out.println(JACK.name);
JACK.displayEmail( );   // call a function

Message ANDY = new Message ("ANDY", "ANDY@xxx.xxx", "567-xxx-0000");   //create an object ANDY, and call constructor
System.out.println(ANDY.name);
ANDY.displayPhone( );    // call a function
}
}
```

Click White Triangle Green Button in tool bar to run the program and see the output:

Output:

```
JACK

Email:JACK@xxx.xxx

ANDY

Phone:567-xxx-0000
```

Explanation:

“public class Message” defines a class “Message”.

“public Message(……)” defines a constructor, initializes three string variables.

“ Message JACK = new Message (……)” creates an object “JACK”, and automatically calls the constructor to initialize three string variables.

“JACK.name” references the variable “name”.

“JACK.displayEmail();” calls the function “displayEmail( )”.

“Message ANDY = new Message (……)”creates an object “ANDY”, and automatically calls the constructor to initialize three string variables.

“ANDY.name” references the variable “name”.

“ANDY.displayPhone();” calls the function “displayPhone( )”.

Hour 6

Inheritance & Modifiers

Inheritance

A class can be extended by a subclass.

```
class sub-class extends parent-class;
```

“extends” is used in a sub class extends a parent class.

Example 6.1

```
class parent_class{   // parent class
// define variable here;
// define method here;
}
class sub_class extends parent_class{   // sub class
// define variable here;
// define method here;
}
```

Explanation:

“class sub_class **extends** parent_class” means that a sub class extends a parent class (also called super class).

Sub class can access the variable and method in parent class, but cannot access the private variable and private method in parent class.

“super” keyword

In sub class, “super” is used to call the constructor of parent class.

```
super ( argument );
```

“super (argument);” is placed the first line in the constructor of sub class, to call the constructor of parent class.

Example 6.2

```
class super_class{
super_class (int x, int y){   // constructor
this.x=x;  this .y=y;
}}
class sub_class extends super_class{
sub_class(int x, int y){   // constructor
super ( x,y );   // call the constructor of parent class
}}
```

Explanation:

“super (x,y);” calls the constructor in super class.

Overriding

The method of sub class can override the method of super class, if the method name and argument are the same.

Example 6.3

```
class Super_class{
int test( int num ) { return num; }   // overriding
}
class Sub_class extends Super_class{
int test( int num ) { return 100 + num; }   // overriding
}
public class OverridingClass {
public static void main (String args[ ]){
Sub_class obj=new Sub_class( );
System.out.print( obj.test( 200 ));
}}
```

(Output: 300)

Explanation:

When “obj.test(200)” calls the “int test( ){ }”, the “int test( ){ }” in sub class overrides the “int test( ){ }” in super class, because they have the same name and same argument.

Overloading & Overriding

overloading	same method name, different argument, in the same class.
overriding	same method name, same argument, in between super class and sub class.

Example 6.4 (overloading)

class Color{ String x,y;

Color (String a, String b) { x=a; y=b; }   // overloading

Color ( ) { x=”yellow”; y=”purple”;} // overloading

}

Example 6.5 (overriding)

```
class super_class{
int test( int num ) { return num; }    // overriding
}
class sub_class extends super_class{
int test( int num ) { return 100 + num; }   // overriding
}
```

Explanation:

Example 1 is overloading. Example 2 is overriding.

Static variable

“static variable” is known as class variable, it can be referenced by the class or object.

“Non static variable” cannot be referenced by class.

“Non static variable” can be referenced by object only.

Example 6.6

```
class MyClass {
static int count=100;   // declare a static vaiable
public static void main (String args[ ]){
System.out.print(  MyClass.count );   /* MyClass references a static variable
“count” directly  */
}}       (  Output:  100 )
```

Explanation:

“static int count=100” defines a static variable.

“MyClass.count” means a class “MyClass” references a static variable “count” directly.

Note that an object can also reference static variable.

Static Method

“static method” is known as class method, it can be called by the class or object.

“Non static method” cannot be referenced by class.

“Non static method” can be referenced by object only.

Example 6.7

```
class MyClass {
static int count( ) { return 100 ;}    // declare a static method
public static void main (String args[ ]){
System.out.print(  MyClass.count( ) );  /* MyClass calls a static method “count()” directly  */
}}      ( Output:  100 )
```

Explanation:

“static int count( ) { return 100 }” defines a static method.

“MyClass.count( )” means a class “MyClass” calls a static method “count()” directly.

Note that an object can also call static method.

final Variable

“final” is a keyword.

“final+ variable” means that variable value cannot be modified.

Example 6.8

```
class MyClass{
final int x =100;   // declare a final variable
static void test(  ){ x = 200;}   // error! for "x" is a final variable
public static void main (String args[ ]){
System.out.print(  MyClass.test( ) );
}}
```

Explanation:

“**final** int x =100” defines a final variable “x”, its value is 100 and cannot be changed.

“static void test( ) { x = 200;}” tries to modify the value of “x”, but error occurs.

final Method( )

“final” is a keyword.

“final method( )” means that method cannot be overridden.

Example 6.9

```
class animal{
final void test ( ){   // declare a final method
System.out.print(“animal”);
}}
class dog extends animal {
void test ( ){    // error! for test() want to override…
System.out.print(“dog”);
}}
```

Explanation:

“**final** void test( ){…}” defines a final method.

“**void test** ( ){…}” tries to override “**final** void test( ){…}”, but error occurs.

final class

“final” is a keyword.

“final class” means that class cannot be extended.

Example 6.10

```
final class animal {   // declare a final class
//  define some variables;
//  define some method;
}
class elephant extends animal {   // error! cannot extends
//  define some variables;
//  define some method;
}
```

Explanation:

“**final** class animal {… }” defines a final class “animal”, which means “animal” class cannot be extended.

“class elephant **extends** animal { …}” tries to extend super class “animal”, but error occurs.

Polymorphism

Polymorphism describes the ability to perform different method for different object.

Example 6.11

```
class Animal{    // parent class
void cry( ){
System.out.println("crying......");;
}
}
class Dog extends Animal{   // sub class
void cry( ){      // overriding
System.out.println("Wow, Wow......");
}
}
class Cat extends Animal{   // sub class
void cry( ){    // overriding
System.out.println("Meow, Meow......");
}
}
class PolymorphismDemo{
public static void main(String args[ ]){   // main function
Animal animalVar;
animalVar = new Dog( );   // create a dog object
animalVar.cry( );              // dog object calls cry()
animalVar = new Cat( );    // create a cat object
animalVar.cry( );              // cat object calls cry()
}
}
```

Output:

```
Wow, Wow……
Meow, Meow……
```

Explanation:

“animalVar = new dog( )” creates a dog object, therefore, “animalVar.cry( )” calls a method cry( ) in sub class Dog.

“animalVar = new cat( )” creates a cat object, therefore, “animalVar.cry( )” calls a method cry( ) in sub class Cat.

Package & Import

```
package package-name;
import package.class;
```

“package package-name” creates or use a package.
“import package.class” imports a package or its class.

Example 6.12 (in file1)

```
package myPackage;    /* create a package including two classes  */
class MyClass1 {…}
class MyClass2 {…}
```

Example 6.13 (in file2)

```
package myPackage;    // use package "myPackage"
import myPackage.*;    // import all classes of myPackage
```

Explanation:

“package myPackage” creates a package named “myPackage”, which contains two classes.”MyClass1” and “MyClass2” in file1.java

“import myPackage**.*** ” imports any classes inside the package “myPackage” from file1 to file2.

Note that “package” command must be in above of the “import” command.

Build a Package, Import a Class

```
package myPackage;     // build a package
import anyClass;     // import a class
```

Example 6.14

(In Drive.java)

```
package myPackage;   // create a package containing a class
public class Drive{
public void getDistance (int mph, float time){
System.out.println(“SPEED:” + mph + “ MILES/HOUR”);
System.out.println(“TIME:” + time + “ HOURS”);
System.out.println(“DISTANCE:” + time*mph + “ MILES”);
}
}
```

(In TestDrive.java)

```
package myPackage;   // use “myPackage”
import myPackage.Drive;    // import class “Drive” in myPackage
public class TestDrive{
  public static void main (String args[ ]) {
            Drive myDrive = new Drive( );
            myDrive.getDistance(120, 5);
}
}
```

Output:

```
SPEED: 120 MILES/HOUR
TIME: 5 HOURS
DISTANCE: 600 MILES
```

Explanation:

“package myPackage;” creates a package “myPackage” in Drive.java file.

“import myPackage.Drive;” imports a class “Drive” of “myPackage”.

“Drive myDrive = new Drive( )” can create an object “myDrive” in file TestDrive, because Drive class has been imported from Drive.java file.

“myDrive.getDistance(120, 5);” calls a method in Drive.java file.

Note: “package” command must be in above of the “import” command.

Hands-On Project: Inheritance

Inheritance Example

Start up Eclipse > Select “myPackage” > new > class > name > InheritanceDemo > Finish.

Write following codes in Eclipse.

```
package myPackage;
class code1{     // parent class
int x, y;
code1(int a, int b){    // constructor
x = a; y = b;
}
}
class code2 extends code1{   // sub class
code2 (int a, int b){    // constructor
super (a,b);   // call the constructor in parent class
}
}
public class InheritanceDemo {

public static void main(String[] args) {   // main function
int result;
code2 number = new code2(100, 200);  /* create an object “number”, and call
code2 constructor automatically */
result = number.x + number.y;
System.out.println("Sum = " + result);
}
}
```

Click White Triangle Green Button in tool bar to run the program and see the output:

Output:

```
Sum = 300
```

Explanation:

“class code1” creates a super class “code1”.

“code1(int a, int b){ }” defines a constructor.

“class code2 extends code1**”** creates sub class “code2” that extends super class “code1”.

“code2 (int a, int b){ }” defines a constructor.

“super (a,b)” calls the constructor in super class so as to initialize the variable x and y.

“public class InheritanceDemo” creates a main class.

“code2 number = new code2(100, 200)” creates an object “number”, automatically calls constructor “code2( ){ }”, and passes two parameters 100 and 200 to constructor, so as to initializes the variable x and y.

“member.x” references variables x, its value is 100.

“member.y” references variables y, its value is 200.

Hour 7

Abstract & Interface

Abstract

Abstract class works as a parent class, which will be extended by its sub class. The method of sub class will override the abstract method of abstract class.

Example 7.1

```
abstract class books{   //define a abstract class
abstract int read( );   //define a abstract method
}
class eBooks extends books{   //extend the parent class
int read ( ){ return 100; }  //override the abstract method
}
```

Explanation:

“abstract class books{ }” defines a abstract class.

“abstract int read( )” defines a abstract method.

“class eBooks extends books { }” means a sub class “eBooks ”extends the parent class ”books”.

“int read ( ) { }” means this method in sub class overrides the abstract method in parent class.

Abstract Example

Example:

```
package myPackage;
abstract class Books{     //create an abstract class
abstract void read();     //create an abstract method
}
class eBooks extends Books{  //extend the parent class
void read(){     //override the abstract method
System.out.println("Overrides the abstract method!");
}}
public class AbstractExample {
public static void main(String[] args){
eBooks obj =new eBooks();
obj.read();
}}
```

Output:

```
Overrides the abstract method!
```

Permission

Java has four permission modifiers to define a member.

default, public, private, protected

Class	default	public	private	protected
same class	V	V	V	V
same package parent class	V	V	X	V
same package sub class	V	V	X	V
different package parent class	X	V	X	X
different package sub class	X	V	X	V

default member cannot be accessed by different package.

public member can be accessed by any package & class.

private member cannot be accessed by different class.

protected member can be accessed by any sub class.

Default Member

default member **cannot** be accessed by **different package**, Note that package name must be the same as directory name where the package locates.

default member has no any modifier.

Example 7.2

There two files in the same package as following:

(In file1 of myPackage)

```
package myPackage;
public class ParentClass{
int num=100;   // This is a default member.
}
```

(In file2 of myPackage)

```
package myPackage;
public class SubClass extends ParentClass{
public static void main(String[] args) {
SubClass s=new SubClass( );
System.out.print ( s.num );  /* s references num in the same package, ok!  */
}
}
```

Output:

```
100
```

Explanation:

“int num=100” defines a default member.
“s.num” can access a default member in the same package.

Public Member

public member can be accessed by any package or any class, spcially in **diffent packages**.

Example 7.3

There two files in the different package as following:

(In file1 of MyPackage)

```
package myPackage;
public class ParentClass{
public int num=100;   // This is a public member.
}
```

(In file2 of NewPackage)

```
package newPackage;
import myPackage.ParentClass;
public class SubClass extends ParentClass{
public static void main(String[] args) {
SubClass s=new SubClass( );
System.out.print ( s.num );  /* s references num in different package, ok!  */
}
}
```

Output:

```
100
```

Explanation:

“public int num=100” defines a public member.
“s.num” can access a public member in different package.

Private Member

private member cannot be accessed by different class.

Example:

```
class MyClass {
private MyClass( ){  // private constructor
System.out.println("This is a private method!");
}}
public class NewClass{
public static void main ( String args [ ] ) {
MyClass obj = new MyClass ( );   /* Error! Access a private constructor  */
}}
```

Output: (Error message!)

Explanation:

“private MyClass( )” defines a private constructor “MyClass”.

“new MyClass ( )” tries to calls private constructor “MyClass”, but error occurs, because they respectively locates different class.

Private Example

private member can be accessed by the same class.

Example:

```
package myPackage;
public class OurClass {
private int a = 100;   // declare a private variable
private int b = 200;   // declare a private variable
public void accessVariable(){
System.out.println("a = " + a);   // access private variable
System.out.println("b = " + b);   // access private variable
}}
public class PrivateExample {
public static void main ( String args [ ] ){
OurClass obj = new OurClass( ); /*  OK!, access a default constructor */
obj.accessVariable();
}}
```

Output:

```
a = 100
b = 200
```

Protected Member

protected member can be accessed any sub class.

Example 7.5

There two files in the different package as following:

(In file1 of MyPackage)

```
package myPackage;
public class ParentClass{   // paraent class
protected int num=100;   // protected member.
}
```

(In file2 of NewPackage)

```
package newPackage;
import myPackage.ParentClass;
public class SubClass extends ParentClass{   // sub class
public static void main(String[] args) {
SubClass s=new SubClass( );
System.out.print ( s.num );  // access a protected member
}
}
```

Output:

```
100
```

Explanation:

“protected int **num**=100” defines a protected member.
“s.**num**” can access a protected member from sub class in different package.
protected member can be accessed any sub class.

Interface

Interface is a special class, which will be implemented by a class. Interface contains one or more empty methods that will be implemented by a method of the class.

Example 7.6

```
interface books {   // define an interface
int booknumber( );    //  define an empty method
}
class eBooks implement books {  //  implement interface
public int booknumber ( ) {  // implement the empty method
return 100;
}  }
```

Explanation:

“interface books” defines an interface named “books”.

“int booknumber( )” defines an empty method “booknumber”.

“class eBooks implement books{   }” means class “eBooks” implements “books” interface.

“public int booknumber( ){   }” implements the booknumber( ).

Interface Example

Example:

```
package myPackage;
interface Books{     //define an interface
public void booksNumbers();   //define an empty method
}
class eBooks implements Books{   //implement interface
public void booksNumbers(){   //implement empty method
System.out.println("Implement the empty method!");
}}
public class InterfaceExample {
public static void main(String[] args) {
eBooks obj = new eBooks();
obj.booksNumbers();
}}
```

Output:

```
Implement the empty method!
```

Abstract & Interface

Abstract Explanation:

```
abstract class books{    //define an abstract class
abstract int read( );   //define an abstract method
}
class eBooks extends books{ //extend the parent class
int read ( ){ return …; } //override the abstract method
}
```

Interface Explanation:

```
interface books {   //define an interface
int booknumber( );    //define an empty method
}
class eBooks implement books {  //implement interface
public int booknumber ( ) {  //implement the empty method
return …;
}
}
```

Initializes Variables

When creating an object, it needs a constructor to initialize the variables.

Example 7.7

```
class Book {  //  create a class
public String title;
public int number;
public Book ( ) {   //  define a constructor
title = “Learning Java ”;  // initialization
number = 100;  // initialization
}}
public class Publisher {   // create main class
public static void main ( String args [ ]) { //main method
Book eBook = new Book ( );  // create an object
System.out.print( eBook.title + eBook.number);
}}
```

(Output: Learning Java 100)

Explanation:

public Book ( ) {   } is a constructor to initialize variables.

Another Class

Main class can use another-class to call another-method.

Example 7.8

```
class AnotherClass{    // define another class
public static void AnotherMethod( ){ // another method
System.out.print ( “Very Good” ); }
}
public class MainClass{  //define main class
public static void main (String args [ ]){ // main method
AnotherClass.AnotherMethod( ); //call another method
}
}
```

(Output: Very Good)

Explanation:

“AnotherClass**.**AnotherMethod( )” use another class to call another method.

Hands-On Project: Max & Min

Abstract Example

Start up Eclipse > Select “myPackage” > new > class > name > AbstractDemo > Finish.

Write following codes in Eclipse.

```
package myPackage;
abstract class A{   // define an abstract class
abstract int max(int x, int y);  // define an abstract method
int min(int x, int y){
return x<y?x:y;
}}
class B extends A{   // extend the parent class
int max(int x, int y){   // override the abstract method
return x>y?x:y;
}}

public class AbstractDemo {
public static void main(String[] args) {
A a;
B b=new B();
int max=b.max(100, 200);
int min=b.min(100, 200);
System.out.println("Max="+max);
System.out.println("Min=" +min);
}}
```

Click White Triangle Green Button in tool bar to run the program and see the output:

Output:

```
Max = 200
Min = 100
```

Explanation:

“abstract class A{   }” defines a abstract class.

“abstract int max( )” defines a abstract method.

“class B extends A {   }” means a sub class “B” extends the parent class ”A”.

“int max ( ) {   }” means this method in sub class overrides the abstract method in parent class.

Hands-On Project: Max & Min

Interface Example

Start up Eclipse > Select “myPackage” > new > class > name > InterfaceDemo > Finish.

Write following codes in Eclipse.

```
package myPackage;
interface myInterface{     // define an interface
public void compareNumbers(int num1, int num2);   /* define an empty method  */
}
class MaxNum implements myInterface{  //implement interface
public void compareNumbers(int num1, int num2){  /* implement the empty method */
System.out.println("Two numbers are: "+num1+" & "+num2);
if (num1>=num2)
System.out.println("Maxmum is:"+num1);
else
System.out.println("Maxmum is:"+num2);
}
}
class MinNum implements myInterface{
public void compareNumbers(int num1, int num2){
System.out.println("Two numbers are: "+num1+" & "+num2);
if (num1<=num2)
System.out.println("Minmum is:"+num1);
else
System.out.println("Minmum is:"+num2);
}
}
public class InterfaceDemo {
public static void main(String[] args) {
MaxNum maxObject = new MaxNum();
maxObject.compareNumbers(100, 120);
MinNum minObject = new MinNum();
minObject.compareNumbers(60, 80);
}
}
```

Click White Triangle Green Button in tool bar to run the program and see the output:

Output:

```
Two numbers are: 100 & 120

Maxmum is:120

Two numbers are: 60 & 80

Minmum is:60
```

Explanation:

“interface myInterface” defines an interface named “myInterface”.

“void compareNumbers ( )” defines an empty method “void compareNumbers( )”.

“class MaxNum implements myInterface { }” means class “MaxNum” implements “myInterface” interface.

“class MinNum implements myInterface { }” means class “MinNum” implements “myInterface” interface.

“public void compareNumbers( ){ }” implements the empty method compareNumbers( ) in interface.

Hour 8

Exception & I/O

Exception

An exception is an error that occurs during the running of a program that breaks the normal flow of commands.

Example 8.1

```
public class ExceptionSample {
public static void main (String args [ ]) {
int a=10, b=0, c;
c = a/b;   // error! because "b" is 0.
System.out.print ( c );
}}
( Output:  java.lang. ArithmeticException:  / by zero)
```

Explanation:

"c = a/b" occurs error, because "b" is zero.

Above example shows an exception whose pattern is "ArthmeticException".

Catch Exception (1)

```
try  {  // some codes that may have exception }
catch ( Exception ) {    // process the exception }
```

“try-catch” can catch exception and handles it.

Example 8.2

```
public class ExceptionSample {
public static void main (String args [ ]) {
try {   // some codes that may have exception
int a=10, b=0, c;
c = a/b;   // error! because “b” is 0.
System.out.print ( c );
}
catch ( ArithmeticException e) {   // handle the exception
System.out.print (" Divided By Zero ");
}}}   ( Output:  Divided By Zero )
```

Explanation:

“try{…}” has “c=a/b”, and causes an exception.

“catch {…}” processs that exception.

Catch Exception (2)

```
try  {  // some codes that may have exception }
catch ( Exception ) { // process exception }
```

When an array index is out bound, exception will occur.

Example 8.3

```
public class ExceptionSample {
public static void main (String args [ ]) {
try {
int a[ ] = new int [10];   // the last index is 9
a[10]=100;  // error! because the index is out bound.
System.out.print ( a[10] );
}
catch ( ArrayIndexOutOfBoundsException e) {
System.out.print (" Array index out of bounds ");
}}}    ( Output:  Array index out of bounds )
```

Explanation:

“try{…}” has “a[10]”, and causes an exception.

“catch {…}” processes that exception.

Catch Exception (3)

```
try  {  // some codes that may have exception }
catch ( Exception ) { // process exception }
```

When a character index is out bound, exception will occur.

Example 8.4

```
public class ExceptionSample {
public static void main (String args [ ]) {
try {
char ch = "ABC".charAt(100);  // error! index out bound.
System.out.print ( ch );
}
catch (IndexOutOfBoundsException e) {
System.out.print ( "Character index out of bounds" );
}}}   ( Output:  Character index out of bounds.)
```

Explanation:

“try{…}” has “ABC”.charAt(100); and causes an exception, because the length of “ABC” is 3, and without the 100th character.

“catch {…}” processes that exception.

Finally Command

The try-catch block can be followed by a “finally” command which always be executed at last.

```
try  {  // some codes that may have exception }
catch ( Exception ) { // process exception }
finally  { // must execute at last  }
```

Example 8.5

```
public class ExceptionSample {
public static void main (String args [ ]) {
try {
char ch = "ABC".charAt(100);  // error!
}
catch (IndexOutOfBoundsException e) {
System.out.print ( "Character index out of bounds. " );
}     (  Output: Character index out of bounds )
finally {   System.out.print("The End"); }  // must execute
}}    ( Output: The End )
```

Explanation:

“finally {   System.out.print(“The End”); }” must be executed.

Throw Exception

Any program can be ready to throw its own exceptions manually.

```
throw new exception ( “exception message”);
```

“throw” is used to throw an exception and error message manually.
“e.getMessage( )” can display error message.

Example 8.6

```
public class ExceptionSample {
public static void main (String args [ ]) {
try {
int a[ ] = new int[10];  // the last element is a[9], without a[10].
if ( a[10] >0 ) throw new ArrayIndexOutOfBoundsException("Error");   // throw
}
catch ( ArrayIndexOutOfBoundsException e) {
System.out.print ("Error index:"+e.getMessage( ));  // alert
}}}     ( Output:  Error index:  10 )
```

Explanation:

“throw new exception(“…”)” can throw an exception.

Throws Exception

Any method can be ready to throw its own exceptions manually. “e.getMessage( )” displays the error message.

```
method ( ) throws exception {…}
```

Example 8.7

```
public class ExceptionSample {
public static void main (String args [ ]) {
try {
int b = 0;
calculate( b );   //call the function, but exception occurs!
}
catch( Exception e) {
System.out.println("The error is"+ e.getMessage( ));
}}
public static int calculate( int num) throws ArithmeticException {     // throws
exception manually
int a=10, c;
c = a/num;   // num is 0, error!
return c;
}
}
```

(Output: The error is: / by zero.)

Explanation:

“int calculate(int num) **throws** ArithmeticException” throws an exception.

“e.getMessage( )” displays the error message.

File Class

File class is used to display the property of file, to check the file existence, writable, readable, to rename the file……

```
File f = new File (file-name);     // create a file object
f.exists();              //  check existence
f.canRead( );       //  check readable
f.canWrite( );       //  check writable
f.isfile( );              //  check if is a file
```

Example 8.8

```
File f = new File ( MyFile );
if (f.exists( )) {
System.out.println("Exists:"+ f.exists());     // Exists: true
System.out.println("Can read:"+ f.canRead());  // Can read: true
System.out.println("Can write:"+ f.canWrite());  // Can write: false
System.out.println("Is file:"+ f.isFile());  // Is file: true
}
```

Explanation:

All above method returns true or false.

FileOutputStream

FileOutputStream class is used to process Output stream.

```
FileOutputStream fout = new FileOutputStream (file-
name);

System.in.read( );   //  read inputted characters

fout.write( );      //  write characters to file
```

“new FileIOutputStream (file-name)” creates a Output stream.

“System.in.read( );” reads inputted characters from keyboard.

“fout.write( );” writes characters to the file.

Example 8.9

(Prepare an empty file “**MyFile.txt**” in the same project folder with the following Java file OutputFile.java)

```
package myPackage;
import java.io.*;
public class OutputFile {
public static void main(String[] args) {
char ch;
int number;
try{
FileOutputStream fout = new FileOutputStream("MyFile.txt");
System.out.println("Please input some words, finish with # key and enter:");
while((ch = (char)System.in.read())!= '#')  //read inputed characters from keyboard
fout.write(ch);  // write characters to file
fout.close();
}
catch(FileNotFoundException e){
System.err.println(e);
}
catch(IOException e){
System.err.println(e);
}
}
}
```

(Run the program, and type “C++ is very good!” in keyboard)

Result:

Please check the MyFile.txt, you will find that the content is “C++ is very good!”.

Explanation:

“fout” is an output stream object.

“System.in.read( )” reads characters from keyboard.

“fout.write(ch)” writes characters to the file.

When you view the MyFile.txt, its content is:

“C++ is very good”

FileInputStream

FileInputStream class is used to process input stream.

```
FileInputStream fin = new FileInputStream (file-
name);

fin.read( );      //  reads content of a file
```

"new FileInputStream (file-name)" creates an input stream.

"fin.read( );" reads content of a file.

Example 8.10

(Given **MyFile.txt** content is “C++ is very good!” in the same project folder with following Java file InputFile.java)

```
package myPackage;
import java.io.*;
public class InputFile {
public static void main(String[] args) {
char ch;
int number;
try{
FileInputStream fin = new FileInputStream("MyFile.txt");
while((number=fin.read())!= -1)  // read the file
System.out.print((char)number);  // change to characters
fin.close();
}
catch(FileNotFoundException e){
System.err.println(e);
}
catch(IOException e){
System.err.println(e);
}}}
```

Output:

```
C++ is very good!
```

Explanation:

“fin” is an input stream object.

“fin.read()” reads the content of MyFile.txt

“-1” indicates the end of file.

Create Thread

Threads can be used to perform multiple tasks simultaneously. Each thread represents a subtask. There are two methods to create a thread.

Method One:

```
public class ThreadDemo extends Thread{  //
extends
public void run( ){
// the thread body
}
}
public static void main(String[ ] args){
ThreadDemo myThread = new ThreadDemo( );
myThread( ).start( );    // start thread
}
```

Explanation:

The method one uses **“extends Thread”** class.

“run( )” automatically runs when thread start.

“myThread = new ThreadDemo( );” creates a thread object “myThread”.

“start( )” starts the thread.

Method two:

```
class ThreadDemo implements Runnable{  //
implements
public void run( ){
// the thread body
}
}
public static void main(String[ ] args){
Thread myThread = new Thread(new
ThreadDemo( ));
myThread.start( );    // start thread
}
```

Explanation:

The method two uses **"implements Runnable"** interface.

"run( )" automatically runs when thread start.

"new Thread(new ThreadDemo( ))**"** creates a thread object "myThread".

"start( )" starts the thread.

Extends Thread

Example 8.11

```
public class ThreadDemo extends Thread{  // extends
public void run( ){      // thread body
for(int i = 0; i<3; i++){
System.out.println(“Thread!”);
}}
public static void main(String[ ] args){
ThreadDemo myThread = new ThreadDemo( );
myThread.start( );      // start thread
}}
```

Output:

```
Thread!
Thread!
Thread!
```

Explanation:

“run( )” automatically runs when thread start.

“myThread = new ThreadDemo( );” creates a thread object “myThread”.

“start( )” starts the thread.

Implements Runnable

Example 8.12

```
class ThreadDemo implements Runnable{ // implements
public void run( ){     // thread body
for(int i = 0; i<3; i++){
System.out.println(“Thread!”);}
}
public static void main(String[ ] args){
Thread myThread = new Thread(new ThreadDemo( ));
myThread.start( );    // start thread
}}
```

Output:

```
Thread!
Thread!
Thread!
```

Explanation:

“run( )” automatically runs when thread start.

“myThread = new Thread(new ThreadDemo( ))” creates a thread object “myThread”.

“start( )” starts the thread.

Multi-Thread

Example 8.13

```
class ThreadDemo extends Thread{
public ThreadDemo(String threadName){
super(threadName);  // initialize thread name
}
public void run( ){   // thread body
for(int i = 0; i<2; i++){
System.out.println(getName( ) + " running!");
}}}
public class UseThread{
public static void main(String[ ] args){
ThreadDemo myThread1 = new ThreadDemo("1st thread");
ThreadDemo myThread2 = new ThreadDemo("2nd thread");
ThreadDemo myThread3 = new ThreadDemo("3rd thread");
myThread1.start();    // start thread
myThread2.start();   // start thread
myThread3.start();    // start thread
}}
```

Output:

```
1st thread running!
1st thread running!
2nd thread running!
2nd thread running!
3rd thread running!
3rd thread running!
```

Explanation:

“super(threadName)” initializes the thread name by Thread class constructor.

“getName( )” returns thread name.

“run( )” automatically runs when thread start.

“new myThread(“1st thread”)” creates a thread object named “1st Thread”.

“myThread1.start();” starts the thread “myThread1”.

……

Thread’s Methods

thread.start( ) starts the thread.

thread.stop( ) stops the thread.

thread.sleep( ) makes the thread sleep for some time.

thread.wait( ) makes the thread in waiting status.

thread.notify( ) wakes up the thread that is waiting.

thread.yield( ) lets other threads with same priority run.

thread.isAlive( ) tests the current thread alive.

thread.currentThread( ) gets the current thread.

thread.interrupt( ) stops the “blocked” thread.

thread.suspend( ) makes the thread pause.

thread.resume( ) starts the thread again.

thread.join( ) runs this thread after other thread ended.

For example:

```
if(time_to_stop){
thread.currentThread( ).stop( );  // stop current thread.
}
```

Thread’s Example

Example 8.14

```
class ThreadDemo implements Runnable{
public void run( ){   // thread body
……
wait( );   // make the thread in waiting status.
……
}
}
Class TextThread{
public static void main(String args[ ]){
Thread myThread = new Thread(newThreadDemo( ));
myThread.start( );   // start thread
Thread.sleep(1000);   // make the thread sleep for 1 second
notify( );  // wake up the thread that is waiting
Thread.yield( );   /*  let other threads with the same priority run first. */
}
}
```

Explanation:

“wait( )” makes the current thread waiting.

“new Thread(newThreadDemo( )” creates a thread.

“myThread.start( )” starts the thread.

“Thread.sleep(1000)” makes the thread sleep for 1000 milliseconds.

“notify( )” wakes up the thread that is waiting.

“Thread.yield( )” lets other threads with the same priority run first.

Note: suspend ( ) and resume ( ) are not recommended to use nowadays.

Hands-On Project: Out of Range

Exception Example

Start up Eclipse > Select “myPackage” > new > class > name > ArrayOutRange > Finish.

Write following codes in Eclipse.

```
package myPackage;
public class ArrayOutRange {
public static void main(String[] args) {
try{  // exception may occur here
int n = 0;
String color[ ]={"Red", "Yellow", "Green"};
while(n < 4){  // if the index equals 3, an exception occurs
System.out.println(color[n]);
n++;
}}
catch(ArrayIndexOutOfBoundsException e){
// process the exception
System.out.println(“     ”);
System.out.println("Exception Occurs!");
System.out.println("Array Index is Out of Range!");
}}}
```

Click White Triangle Green Button in tool bar to run the program and see the output:

Output:

```
Red
Yellow
Green

Exception Occurs!
```

Array Index is Out of Range!

Explanation:

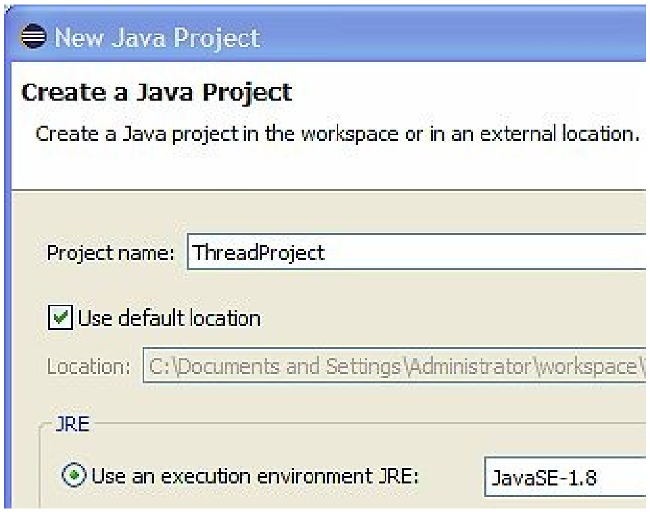
In array “String color[ ]={"Red", "Yellow", "Green"};”, if the index equals 3, an exception occurs.

Hands-On Project: Mult Tasks

Multi-Thread Program

Step 1: Create a Java Project

Start up Eclipse > File > New > Java Project > Project Name > “ThreadProject” > Finish. (Figure 1)



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: ThreadProject

☑ Use default location

Location: C:\Documents and Settings\Administrator\workspace\

JRE

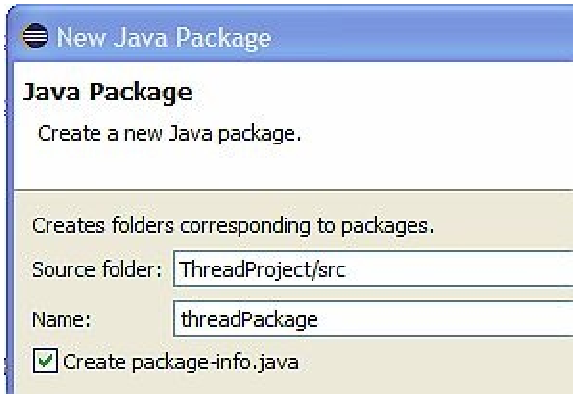
● Use an execution environment JRE: JavaSE-1.8

(Figure 1)

Step 2: Create a Package

Package is used to contain various categories of Java files. Different package contains different Java files.

Select project “ThreadProject” > new > package > name > “threadPackage” > Finish. (Figure 2)



New Java Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

Source folder: ThreadProject/src

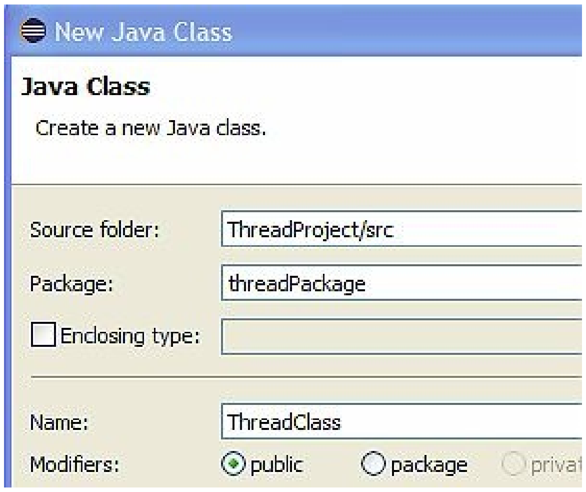
Name: threadPackage

☑ Create package-info.java

(Figure 2)

Step 3: Create a Class

Select package “threadPackage” > new > class > name > “ThreadClass” > Finish. (Figure 3)



Source folder: ThreadProject/src
Package: threadPackage
☐ Enclosing type:
Name: ThreadClass
Modifiers: ● public ○ package ○ privat

(Figure 3)

Step 4: Write Java Codes

Write Java codes in Eclipse.

```
package threadPackage;
public class ThreadClass {
public static void main(String[] args) {   // main function
myThread thread1=new myThread("Thread One");
myThread thread2=new myThread("Thread Two");
thread1.start();   // start thread
thread2.start();   // start thread
}
}

class myThread extends Thread{
public myThread(String str)    // constructor
{super(str);}   // initialize the thread
public void run(){    // thread body
for(int I=0; I<2; I++){
```

```
System.out.println(getName( )+ " is running!");
try{sleep((int)(Math.random()*1500));}
catch(InterruptedException e){}
System.out.println(getName()+ " is completed!");
}
}
}
```

Step 5: Run the Program

Click White Triangle Green Button in tool bar to run the program, you can see the output like this:

Output:

```
Thread Two is running!
Thread One is running!
Thread Two is completed!
Thread Two is running!
Thread One is completed!
Thread One is running!
Thread Two is completed!
Thread One is completed!
```

Explanation:

“thread1.start();” and “thread2.start();” runs in turns.

“sleep((int)(Math.random()*1500));” makes one thread sleep for a moment while another thread is running.

Appendix
Java
Tests & Answers

Tests

Please choose the correct answer

(1)

```
package myPackage;
public class Calculation{
public static void main (String args[ ]){
int a = 100, b = 5;
fill in here c = a/b*10;  // data type of “c”
System.out.print("C = " + c);
}
}
```

A. var   B. string   C. boolean   D. int

(2)

```
package myPackage;
public class WhileStatement {
public static void main(String[ ] args) {
int n = 1;
int sum = 0;
fill in here (n <= 100){    // loop statement
sum += n;
n++;
}
System.out.println("Sum = " + sum);
}
}
```

1. for   B. while   C. do   D. switch

(3)

```
package myPackage;
public class TestArray {
public static void main(String[] args) {
int[]myArray;
myArray = fill in here int[10];   // creates an array object
for (int n = 0; n < myArray.length; n++){
myArray[n] = n;
System.out.print(myArray[n] + "");
}
}
}
```

A. array   B. create   C. new   D. object

(4)

```
package myPackage;
public class TwoStrings {
public static void main(String[] args) {
fill in here s1, s2, s3, s4;   // declares four variables
s1 = "C++ in 8 Hours";
s2 = "C++ in 8 Hours";
s3 = "Python in 8 Hours";
s4 = s2;
System.out.println(s1.equals(s2));
System.out.println(s1.equals(s3));
System.out.println(s3.equals(s4));
}
}
```

A. String B. var C. variable D. void

(5)

```
package myPackage;
public class Message {
String name, email, phone;
public fill in here (String theName, String theEmail, String thePhone){    // creates a constructor
name = theName;
email = theEmail;
phone = thePhone;
}
void displayEmail(){
System.out.println("Email:" + email);
}
void displayPhone(){
System.out.println("Phone:" + phone);
}
public static void main(String[] args) {
Message JACK = new Message ("JACK", "JACK@xxx.xxx", "678-xxx-9999");
System.out.println(JACK.name);
JACK.displayEmail( );

Message ANDY = new Message ("ANDY", "ANDY@xxx.xxx", "567-xxx-0000");
System.out.println(ANDY.name);
ANDY.displayPhone( );
}
}
```

A. constructor   B. function   C. method   D. Message

(6)

```
package myPackage;
class code1{
int x, y;
code1(int a, int b){
x = a; y = b;
}
}
class code2 fill in here code1{   // Inheritance
code2 (int a, int b){
super (a,b);
}
}
```

```
public class InheritanceDemo {
public static void main(String[] args) {
int result;
code2 number = new code2(100, 200);
result = number.x + number.y;
System.out.println("Sum = " + result);
}
}
```

A. inheritance B. extend C. extends D. inherits

(7)

```
fill in here class books{   // define an abstract class
fill in here int read( );   // define an abstract method
}
class eBooks extends books{ // extend the parent class
int read ( ){ return 100; } // implement abstract method
}
```

A. abstract B. interface C. void D. public

(8)

```
fill in here books {   // define an interface
int booknumber( );    //  define an empty method
}
```

```
class eBooks implement books {  //  implement interface
public int booknumber ( ) {  // implement method
return 100;
}
}
```

A. abstract B. interface C. void D. public

(9)

```
package myPackage;
public class ArrayOutRange {
public static void main(String[] args) {
int n = 0;
fill in here color[ ]={"Red", "Yellow", "Green"};
// creates a String array
while(n < 4){
System.out.println(color[n]);
n++;
}
}
}
```

A. String B. var C. variable D. void

(10)

```
package threadPackage;
public class ThreadClass {
public static void main(String[] args) {
myThread thread1=new myThread("Thread One");
myThread thread2=new myThread("Thread Two");
thread1. fill in here ;    // start thread1
thread2. fill in here ;    // start thread2
}
}

class myThread extends Thread{
public myThread(String str)    // constructor
{super(str);}   // initialize the thread
public void run(){
for(int I=0; I<2; I++){
System.out.println(getName( )+ " is running!");
try{ sleep((int)(Math.random()*1500)); }
catch(InterruptedException e){ }
System.out.println(getName()+ " is completed!");
}
}
}
```

A. begin()   B. commence()   C. run()   D. start()

(11)

```
int a=100; int b=200;
String result1 = (a<b) fill in here ”apple” : ”banana”;
```

```
// conditional operator
System.out.print ( result1);
```

1. ==   B. ?   C. !=   D. %=

(12)

```
public class TryClass{
public static void main (String [ ] args){
int number=20;
fill in here ( number ) {   // evaluate the variable “number”
case 10 : System.out.print ("Running case 10");  break;
case 20 : System.out.print ("Running case 20");  break;
case 30 : System.out.print ("Running case 30");  break;
default :  System.out.print ("Running default code");  break;
}}}
```

1. switch   B. while   C. do   D. for

(13)

```
public class TryClass{
public static void main (String [ ] args){
int x = 100;
int y = 200;
System.out.println("Greater number is "+ fill in .max(x, y) );
System.out.println("Less number is "+ fill in .min(x, y) );
```

}} // Mathematics Function

1. Mathematics B. Arith C. Math D. Arithmetic

(14)

StringBuffer sb = **fill in here** **StringBuffer** (“abcde”);
// create a StringBuffer object
sb.insert (2, “123”); // Output: abc123de
sb.append (“456”); // Output: abc123de456
sb.reverse ( ); // Output: 654ed321cba

1. create B. object C. class D. new

(15)

package correctPackage;
class MyClass {
int num;
void test(int num) {
 fill in here .num=num; // a keyword represent the current object
 System.out.println(this.num);
};
public static void main(String[ ] args) {
 MyClass obj = new MyClass( );
 obj.test(10);
}

}

1. this B. MyClass C. object D. super

(16)

class MyClass {

fill in here int count=100; // declare a variable that can be referenced by the class directly

public static void main (String args[ ]){

System.out.print(**MyClass.count**);

}

}

1. final B. static C. dynamic D. abstract

(17)

class MyClass{

fill in here int x =100; // define a variable whose value cannot be changed

static void test( ){ x = 200;} // error

public static void main (String args[ ]){

System.out.print(MyClass.test( ));

}

}

}

}

1. static B. private C. final D. void

(18)

```
public class ExceptionSample {
public static void main (String args [ ]) {
try {
int a=10, b=0, c;
c = a/b;   // error
System.out.print ( c );
}
fill in here ( ArithmeticException e) {   // handle the exception
System.out.print (" Divided By Zero ");
}}}   //  Output:  Divided By Zero
```

1. catch B. handle C. exception D. finally

(19)

```
class MyClass {
fill in here MyClass( ){  // set this constructor that cannot be accessed by different class
System.out.print("OK!");
}}
public class NewClass{
public static void main ( String args [ ] ) {
```

```
MyClass mc=new MyClass ( );   // error
}}
```

1. public   B. default   C. protected   D. private

(20)

```
public class ExceptionSample {
public static void main (String args [ ]) {
try {
int b = 0;
calculate( b );
}
catch( Exception e) {
System.out.println("The error is"+ e.getMessage( ));
}}
public static int calculate( int num) fill in here ArithmeticException {
// throw an exception
int a=10, c;
c = a/num;   // num is 0, error
return c;
}
}
```

1. throw   B. throws   C. exception   D. finally

(21)

```
public class TryClass{
public static void main (String [ ] args){
int sum = 100 + 200;
String str = "We love" fill in here "Java";   //connect two strings
int incre = 10;  incre = ++ incre;
System.out.println ( sum );    // Output:  300
System.out.println ( str );    //  Output:  We love Java.
System.out.println ( incre );   // Output:  11
}}
```

A. connect   B. &   C. join   D. +

(22)

```
public class TryClass{
public static void main (String [ ] args){
int x;
fill in here (x = 0; x <= 5; x++) {
System.out.print( x );
}}}
```

A. if   B. do   C. for   D. while

(23)

```
public class TryClass{
```

public static void main (String [ ] args){
double pi = **Math.PI**;
double ran = **fill in here( );** //creates a random number
System.out.println("The PI value is" + pi);
System.out.println("The random number is" + ran);
}}

A. random B. Math.random C. At.random C. ran

(24)
public class TryClass{
public static void main (String [ ] args){
String str = "This is a string";
int length = **str. fill in here;** //return the length of the string
System.out.println("The size of string is " + length);
}}

A. length() B. size() C. length D.size

(25)
public class fill in here { // define a class
String c1, c2;
Color ( ) {c1="yellow"; c2="purple";} // constructor
void brightness ( ) {
System.*out*.println("This flower is " + c1);

```
System.out.println("That flower is " + c2);
}
public static void main(String[] args) {
Color obj = new Color(  );  //create an object, call constructor
obj.brightness ( );
}}
```

A. MyClass   B. Class   C. NewClass   D. Color

(26)

```
class super_class{
super_class (int x, int y){    // constructor
this.x=x;  this .y=y;
}
}
class sub_class extends super_class{
sub_class(int x, int y){    // constructor
fill in here ( x,y );   //calls the constructor in super class
}
}
```

A. super_class   B. super   C. sub_class   D. sub

(27)

```
abstract class Books{     //create an abstract class
abstract void read();    //create an abstract method
}
class eBooks extends Books{
void fill in here(){     //override the abstract method
System.out.println("Overrides the abstract method!");
}}
public class AbstractExample {
public static void main(String[] args){
eBooks obj =new eBooks();
obj.read();
}}
```

A. Book B. eBooks C. read D. abstract

(28)

```
FileOutputStream fout = new FileOutputStream (file-name);
System.in.read( );   //  read inputted characters
fout. fill in here;     //  write characters to file
```

A. write() B. read() C. writeFile() D. readFile()

(29)

```
class MyClass {
int x, y;
MyClass ( ) {
```

__**fill in here**__ (100, 200);   /* a keyword that represents another same-name constructor */

}

MyClass (int a, int b)

x=a; y=b;

}

A. _construct   B. constructor   C. construct   D. this

(30)

class Super_class{

int test(int num) { return num; }

}

class Sub_class extends Super_class{

int __fill in here__(int num) { return 100 + num; }

}   // overriding

public class OverridingClass {

public static void main (String args[ ]){

Sub_class obj=new Sub_class( );

System.out.print(obj.test(200));

}}

A. myFunction   B. myMethod   C. test   D.myTest

(31)

```
 class ParentClass{
fill in here int num=100;   /* define a member that is only accessible by its sub
class’s object  */
}
public class SubClass extends ParentClass{
public static void main(String[] args) {
SubClass s=new SubClass( );
System.out.print ( s.num );
}
}
```

A. default B. public C. protected D. private

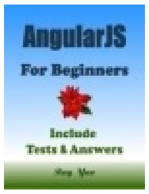
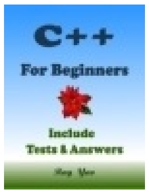
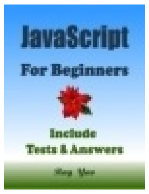
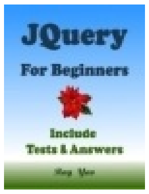
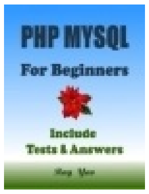
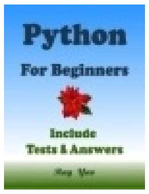
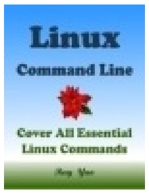
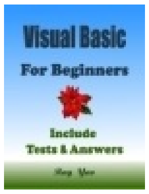
(32)

```
FileInputStream fin = new FileInputStream (file-name);
fin. fill in here( );      //  reads content of a file
```

A. write() B. read() C. writeFile() D. readFile()

Answers

01	D	17	C
02	B	18	A
03	C	19	D
04	A	20	B
05	D	21	D
06	C	22	C
07	A	23	B
08	B	24	A
09	A	25	D
10	D	26	B
11	B	27	C
12	A	28	A
13	C	29	D
14	D	30	C
15	A	31	C
16	B	32	B

 AngularJS For Beginners Include Tests & Answers	C # For Beginners Include Tests & Answers	 C++ For Beginners Include Tests & Answers
HTML CSS For Beginners Include Tests & Answers	JAVA For Beginners Include Tests & Answers	 JavaScript For Beginners Include Tests & Answers
 JQuery For Beginners Include Tests & Answers	 PHP MYSQL For Beginners Include Tests & Answers	 Python For Beginners Include Tests & Answers
 Linux Command Line Cover All Essential Linux Commands	50 PROGRAMS JavaScript For Beginners Learn Coding Fast!	 Visual Basic For Beginners Include Tests & Answers
Advanced C ++ Include Tests & Answers	Advanced JAVA Include Tests & Answers	XML For Beginners Include Tests & Answers

Source Code for Download

Please download the source code of this book:

Java Source Code for Download

Dear My Friend,

If you are not satisfied with this eBook, could you please return the eBook for a refund within seven days of the date of purchase?

If you found any errors in this book, could you please send an email to me? yaowining@yahoo.com

I will be very grateful for your email. Thank you very much!

Sincerely

Ray Yao

My friend, See you!