

JAVA

Overview

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

As of December 2008, the latest release of the Java Standard Edition is 6 (J2SE). With the advancement of Java and its widespread popularity, multiple configurations were built to suite various types of platforms. Ex: J2EE for Enterprise Applications, J2ME for Mobile Applications.

The latest version of Java till date is Java SE 13.0.1 released on September 2019.

Sun Microsystems has renamed the new J2 versions as Java SE, Java EE and Java ME, respectively. Java is guaranteed to be **Write Once, Run Anywhere**.

Features of Java

- **Object Oriented** - In Java, everything is an Object.
- **Platform Independent** - byte code is distributed over the web and interpreted by virtual Machine (JVM)
- **Simple** -
- **Secure** - Authentication techniques are based on public-key encryption.
- Architectural neutral - the compiled code (architecture-neutral object file format) to be executable on many processors
- **Portable** -
- **Robust** - an effort to eliminate error prone situations
- **Multithreaded** - it is possible to write programs that can do many tasks
- **Interpreted** - Java byte code is translated on the fly to native machine instructions
- **High Performance** - With the use of Just-In-Time compilers, Java enables high performance
- **Distributed** - Java is designed for the distributed environment of the internet
- **Dynamic** - Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

The **Java** Development Kit (**JDK**) is a software development environment used for developing **Java** applications and applets. It includes the **Java** Runtime Environment (JRE), an interpreter/loader (**java**), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in **Java** development.

What is JDK , JRE and JVM?

JDK is for development purpose whereas **JRE** is for running the java programs. **JDK and JRE** both contains **JVM** so that we can run our java program. **JVM** is the heart of java programming language and provides platform independence.

A **JAR** (**Java** Archive) is a package **file** format typically used to aggregate many **Java** class **files** and associated metadata and resources (text, images, etc.) into one **file** to distribute application software or libraries on the **Java** platform.

Object - Objects have states and behaviors. Example: A dog has states-color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.

- **Class** - A class can be defined as a template/blue print that describes the behaviors/ states that object of its type support.
- **Methods** - A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** - Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

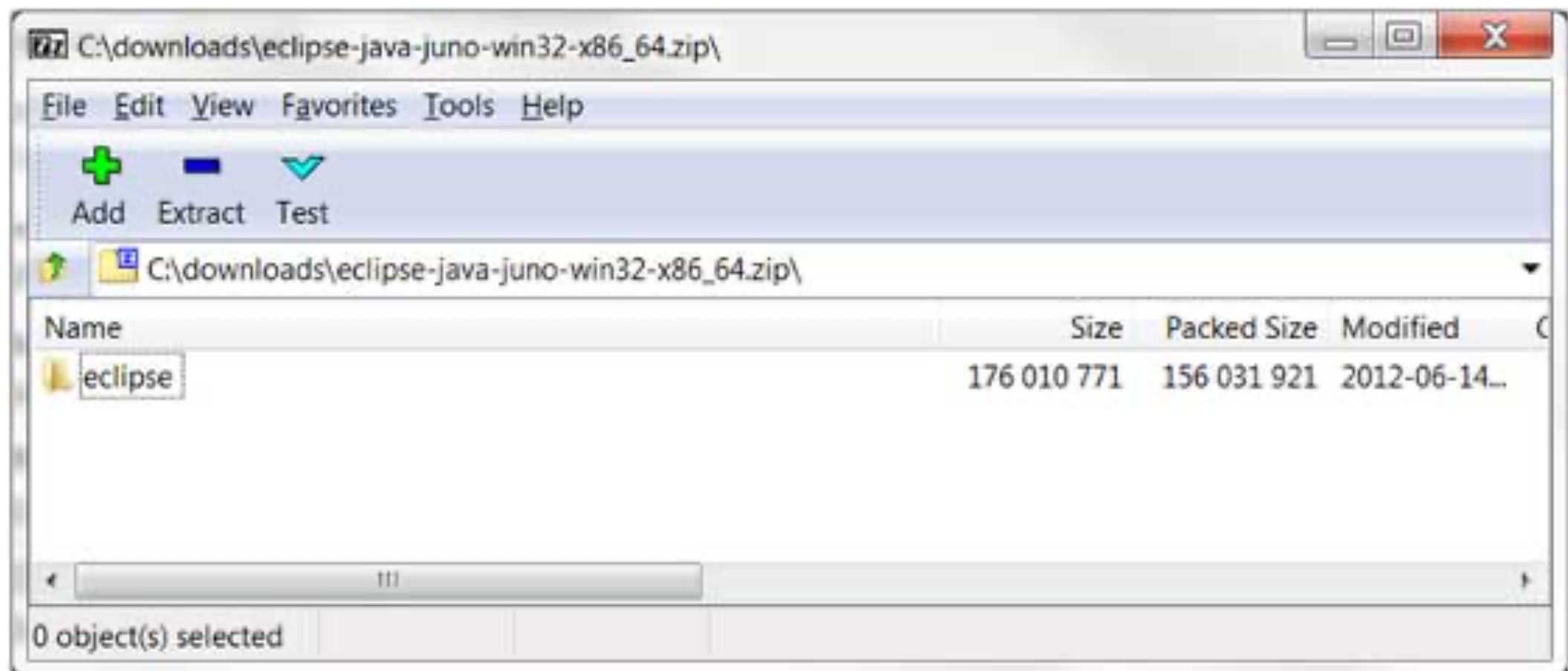
Eclipse:

Eclipse is an integrated development environment (IDE) for developing applications using the Java programming language and other programming languages such as C/C++, Python, PERL, Ruby etc.

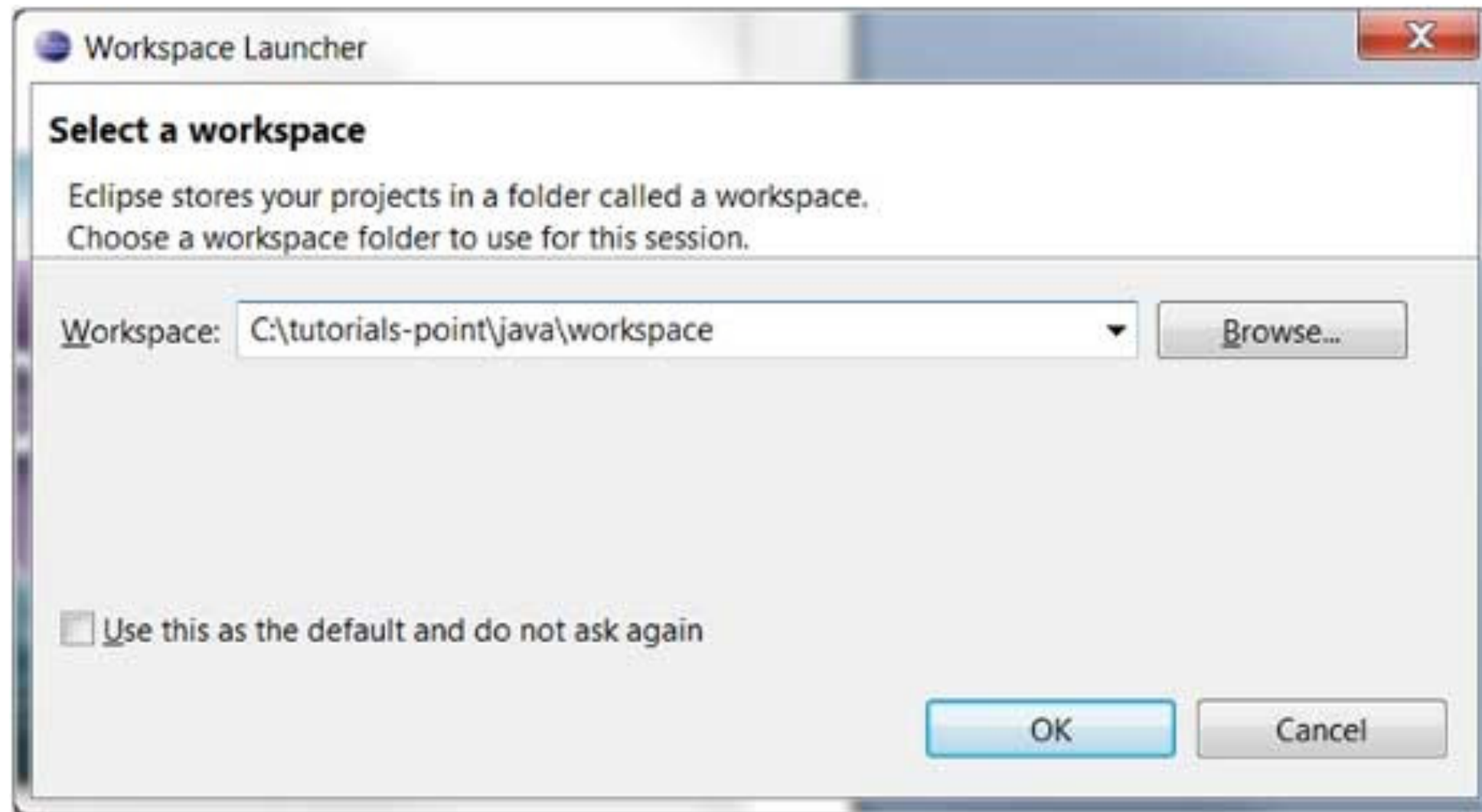
Downloading Eclipse

You can download eclipse from <http://www.eclipse.org/downloads/>

Installing Eclipse



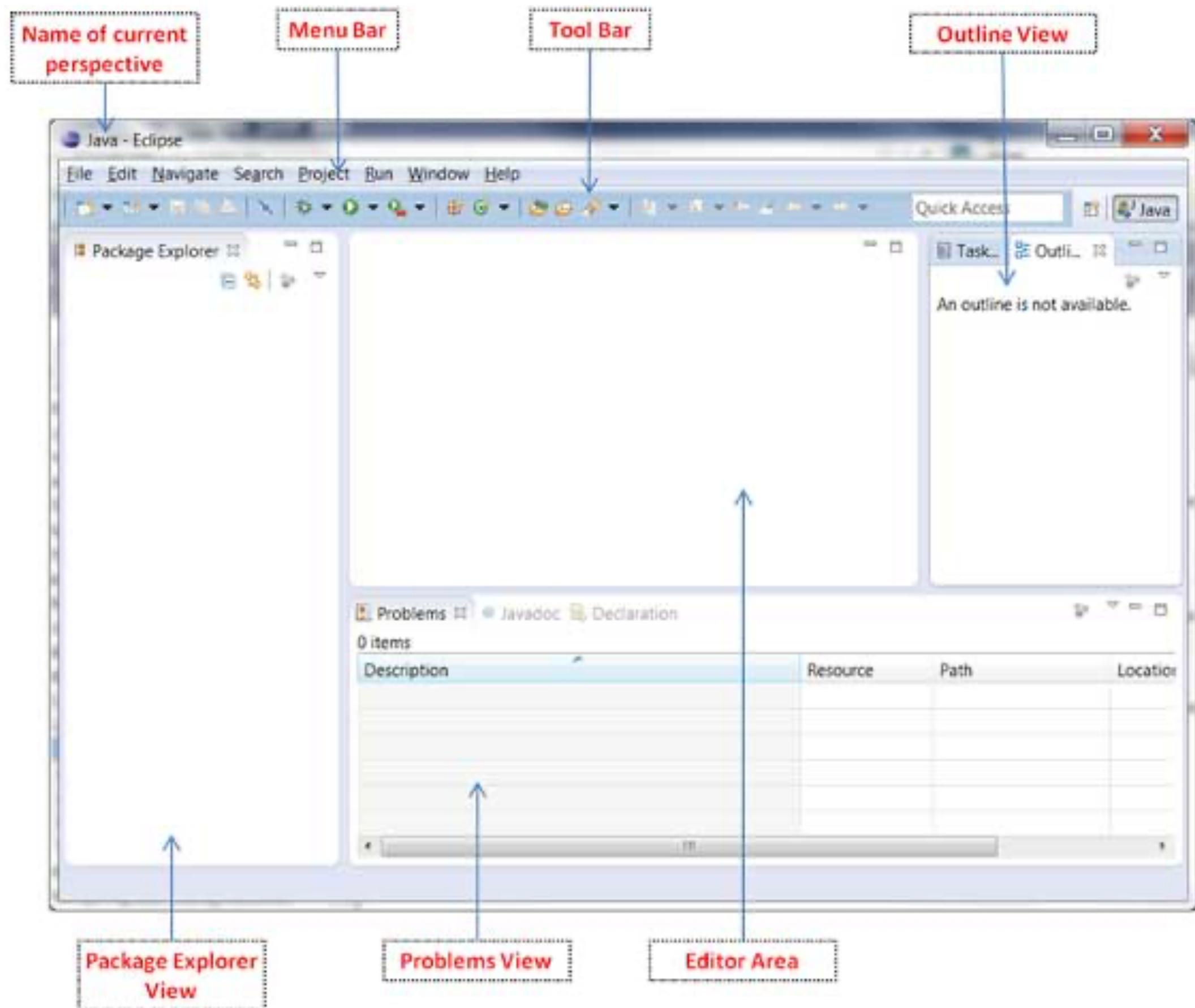
Launching Eclipse:



Parts of an Eclipse Window

The major visible parts of an eclipse window are –

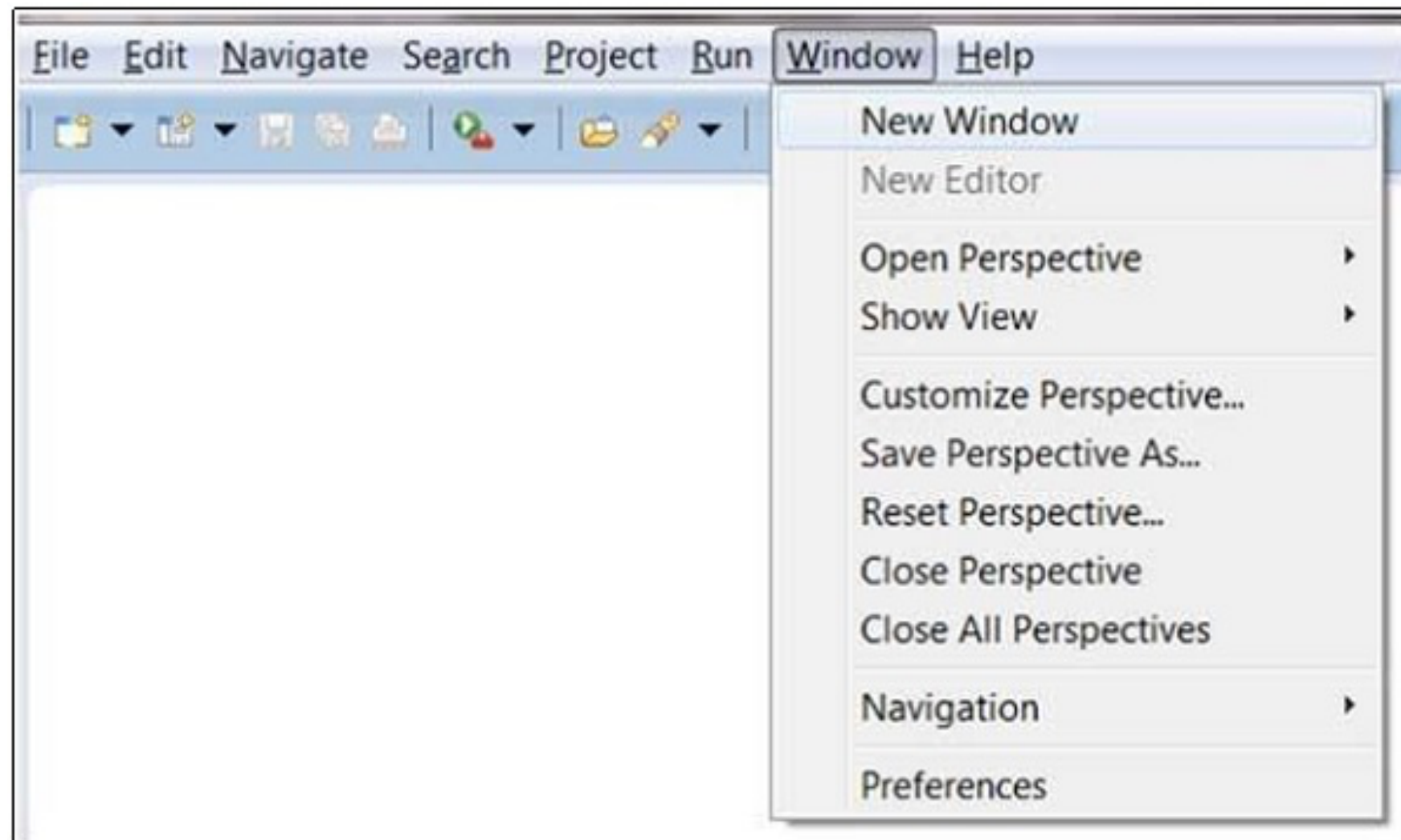
- Views
- Editors (all appear in one editor area)
- Menu Bar
- Toolbar



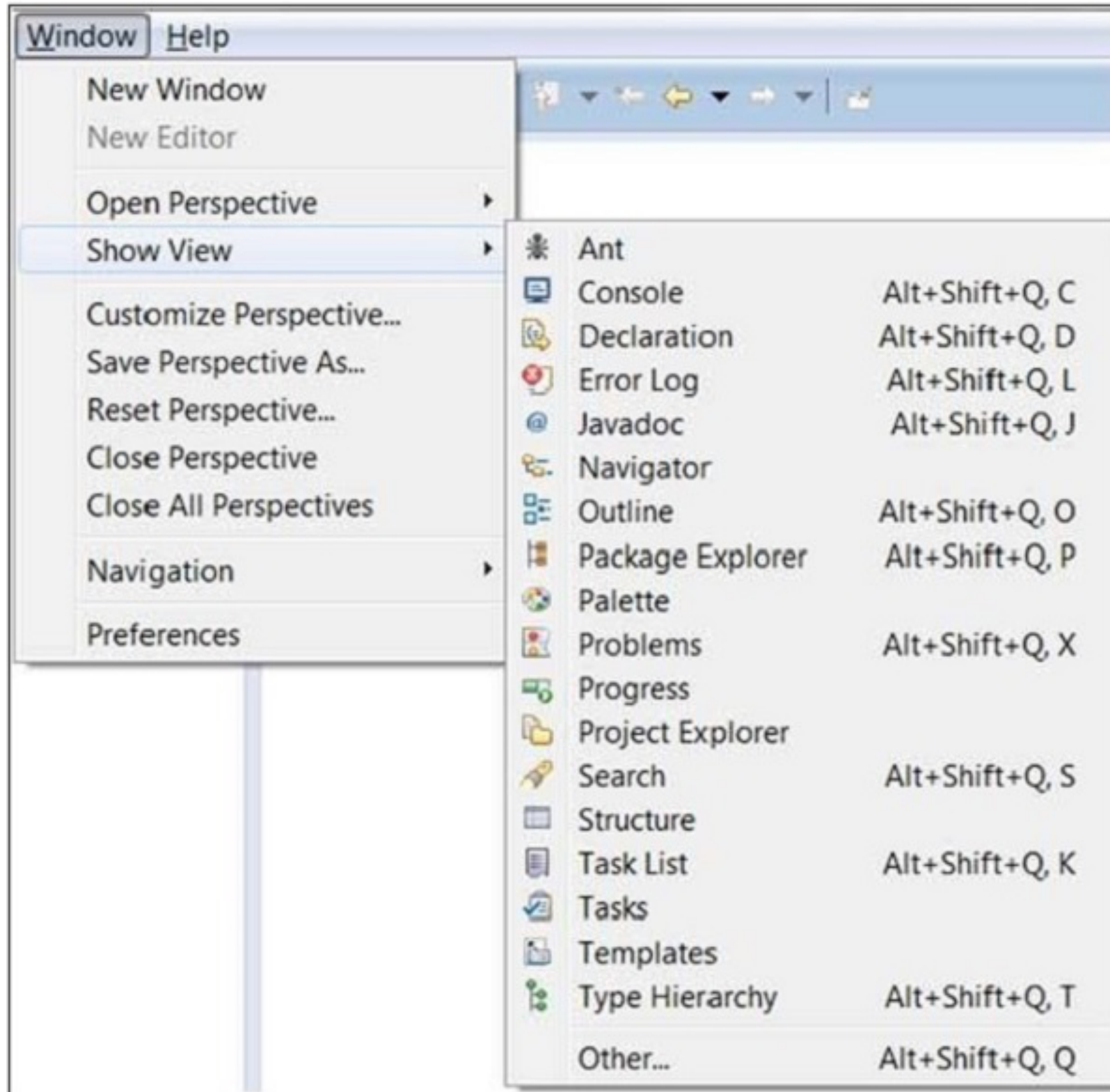
Typical Eclipse Menus

The typical menus available on the menu bar of an Eclipse window are –

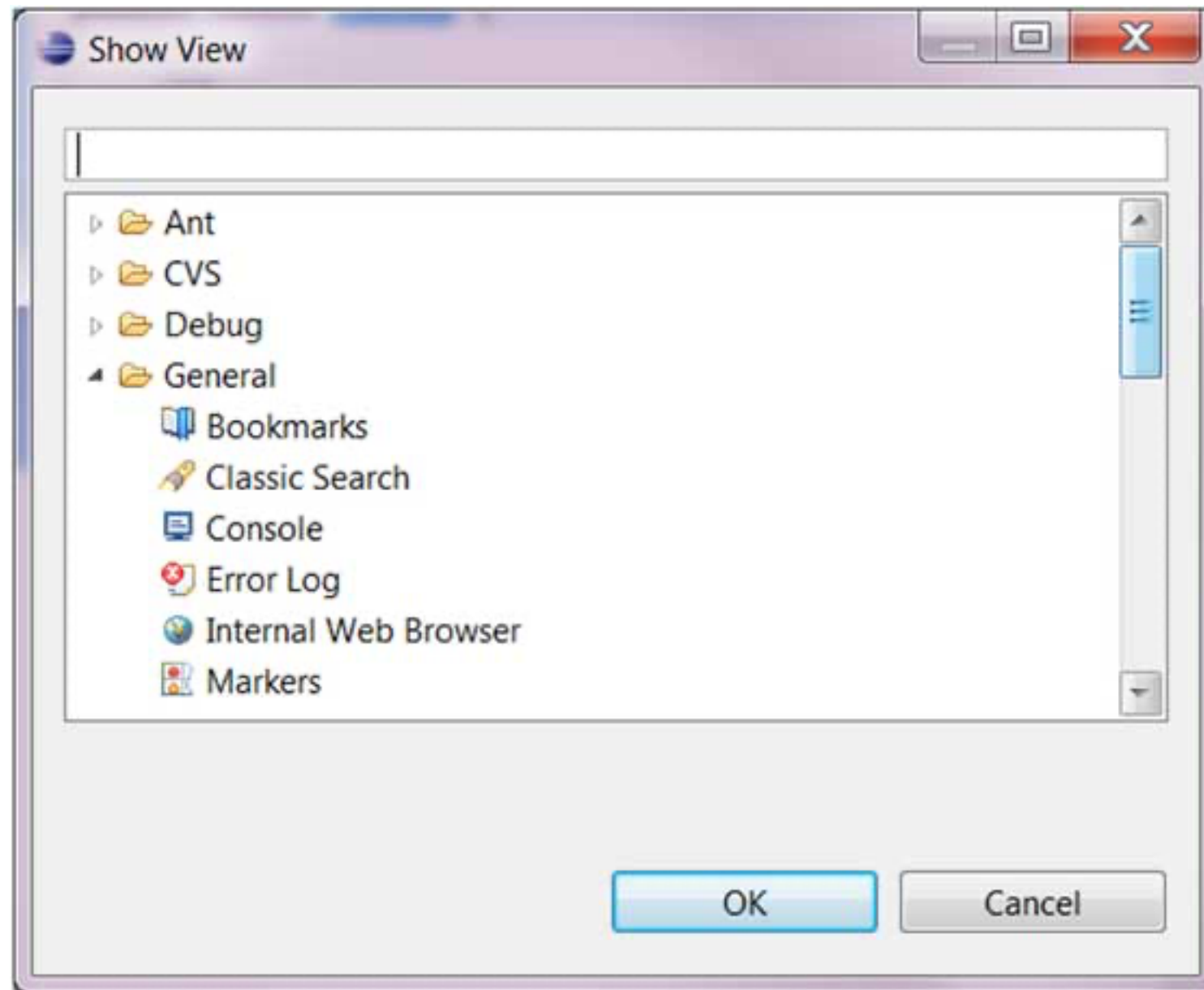
- File menu
- Edit menu
- Navigate menu
- Search menu
- Project menu
- Run menu
- Window menu
- Help menu



Opening a view:



Clicking on the **Other** menu item brings up the Show View dialog box that allows you to locate and activate a view.



What is a Perspective?

An eclipse perspective is the name given to an initial collection and arrangement of views and an editor area. The default perspective is called java. An eclipse window can have multiple perspectives open in it but only one perspective is active at any point of time.

Opening a Perspective

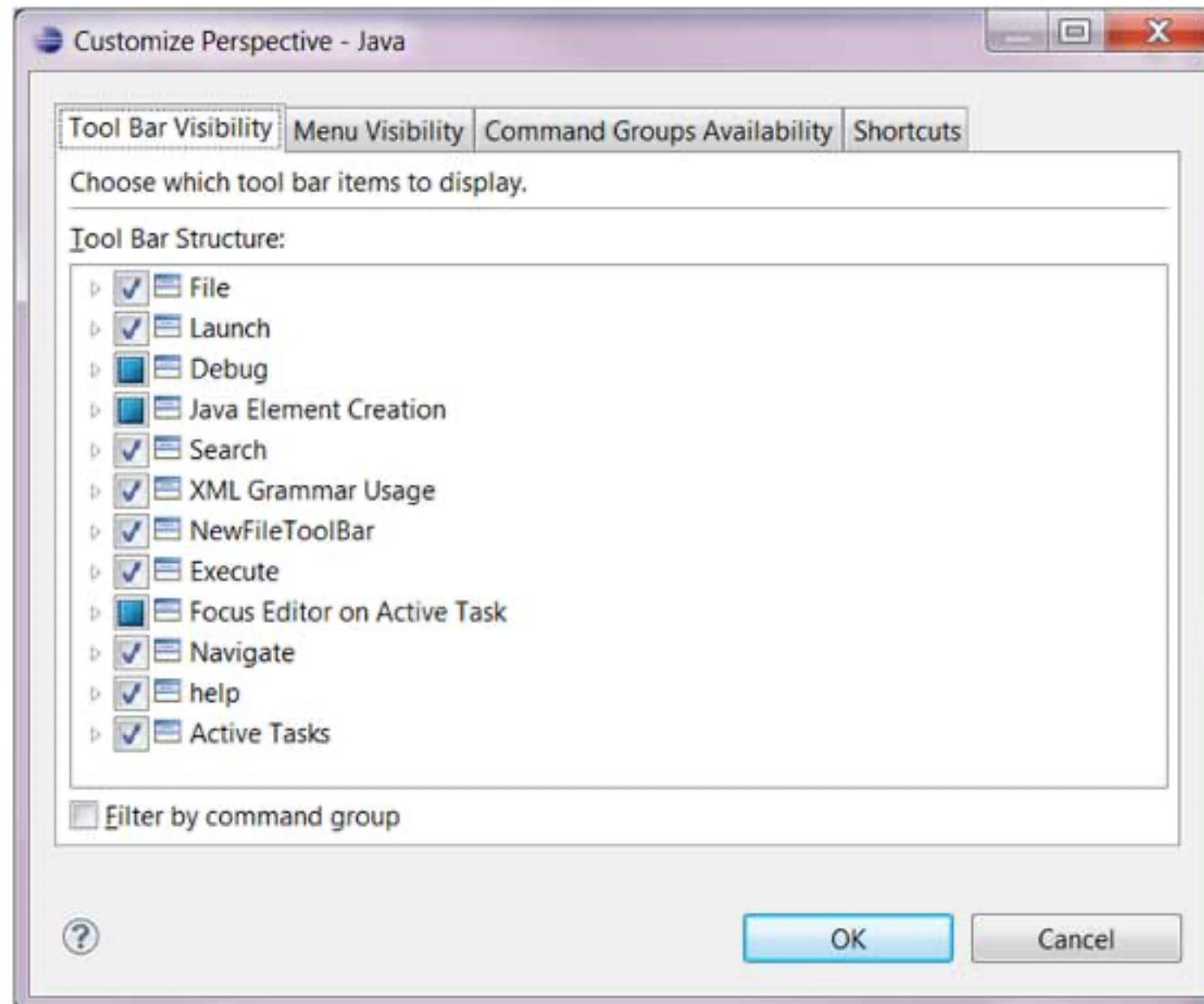
To open a new perspective, click on the Windows menu and select Open Perspective → Other



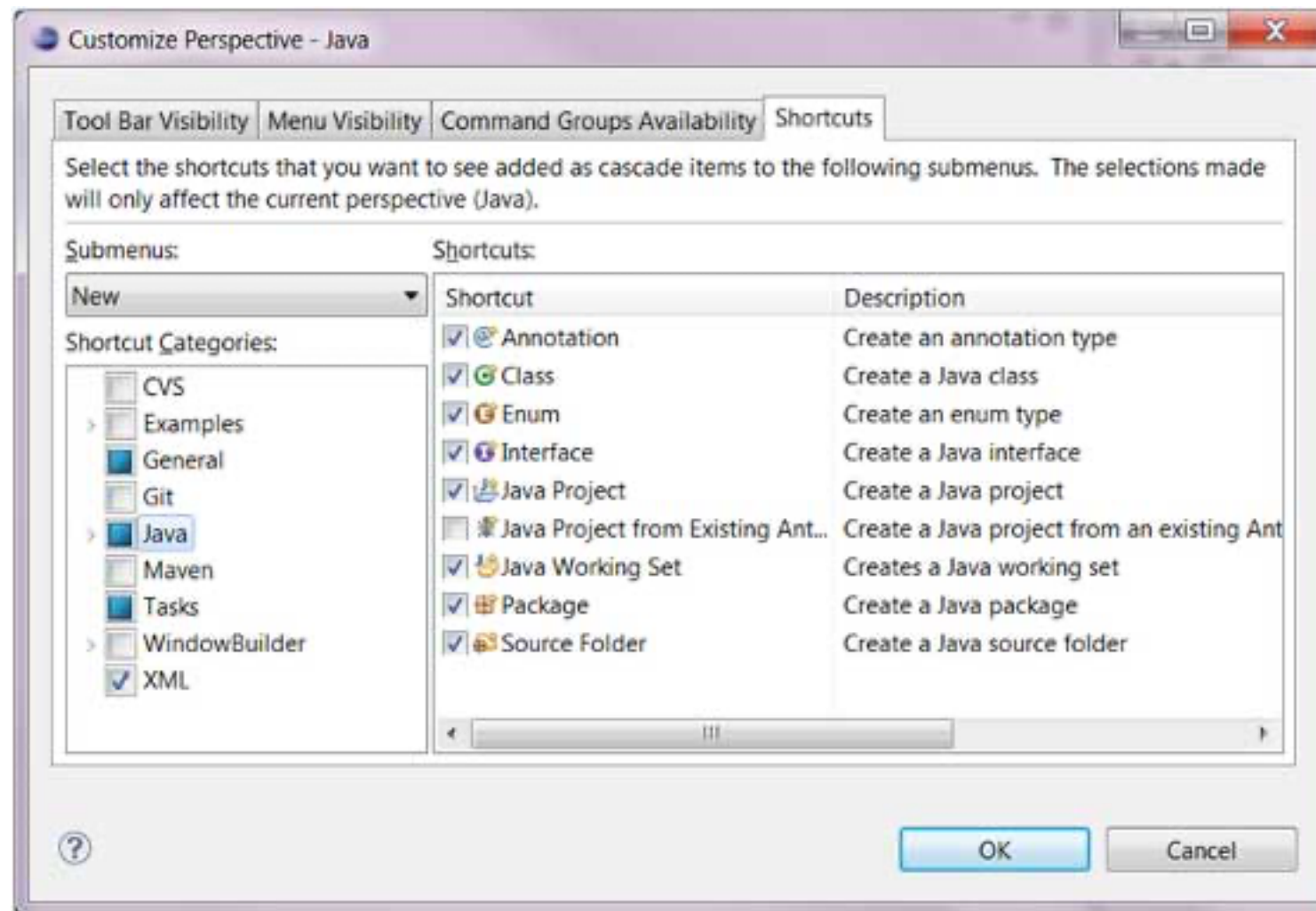
Customizing a Perspective

The customize perspective dialog can be used to customize a perspective. Customizing a perspective means –

- Determining the icons visible on the toolbar when a perspective is active.
- Determining the menu items visible when a perspective is active.
- Determine the menu items in New submenu, Show View submenu and Open Perspective submenu.



- The **Tool Bar Visibility** tab can be used to determine which icons are visible on the toolbar when a perspective is open.
- The **Menu Visibility** tab can be used to determine which menu items are visible when a perspective is active.
- The **Command Groups Availability** tab can be used to control the visibility of toolbar icons and menu items.
- The **Shortcuts** tab can be used to determine the menu items in New submenu, Show View submenu and Open Perspective submenu.

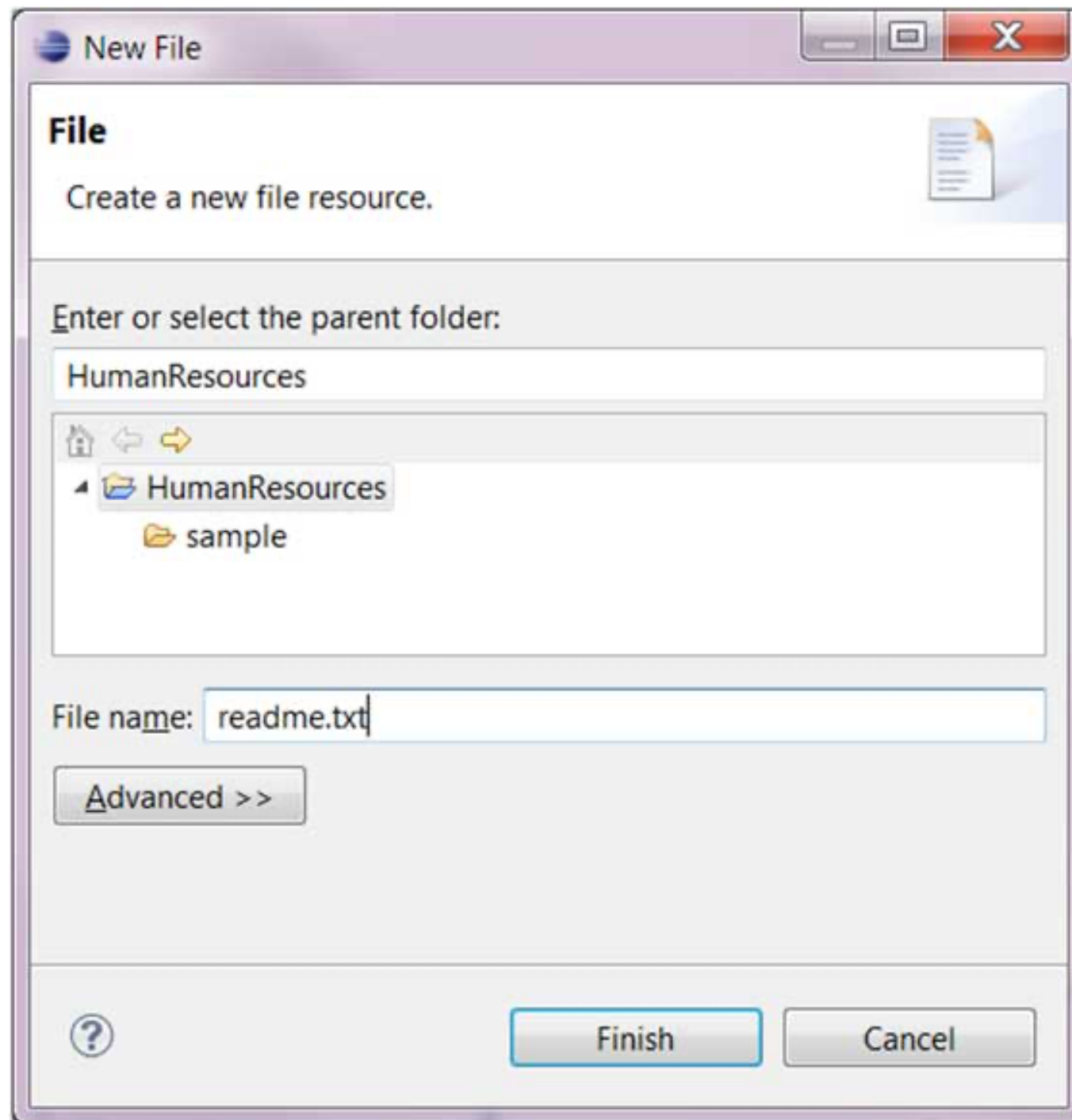


About Eclipse Workspace

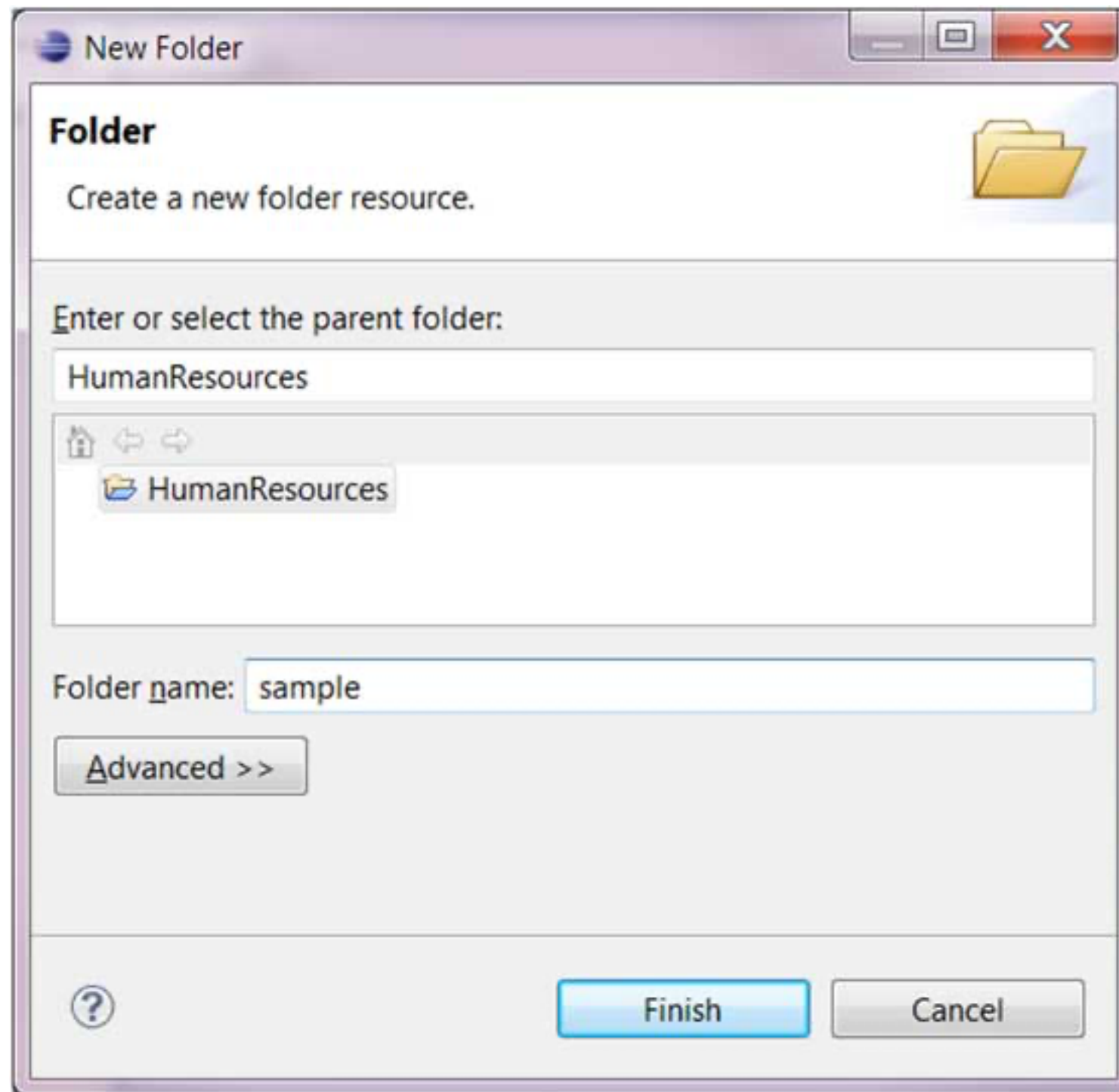
The eclipse workspace contains resources such as –

- Projects
- Files
- Folders


The File Wizard (File → New → File) can be used to create a new file.

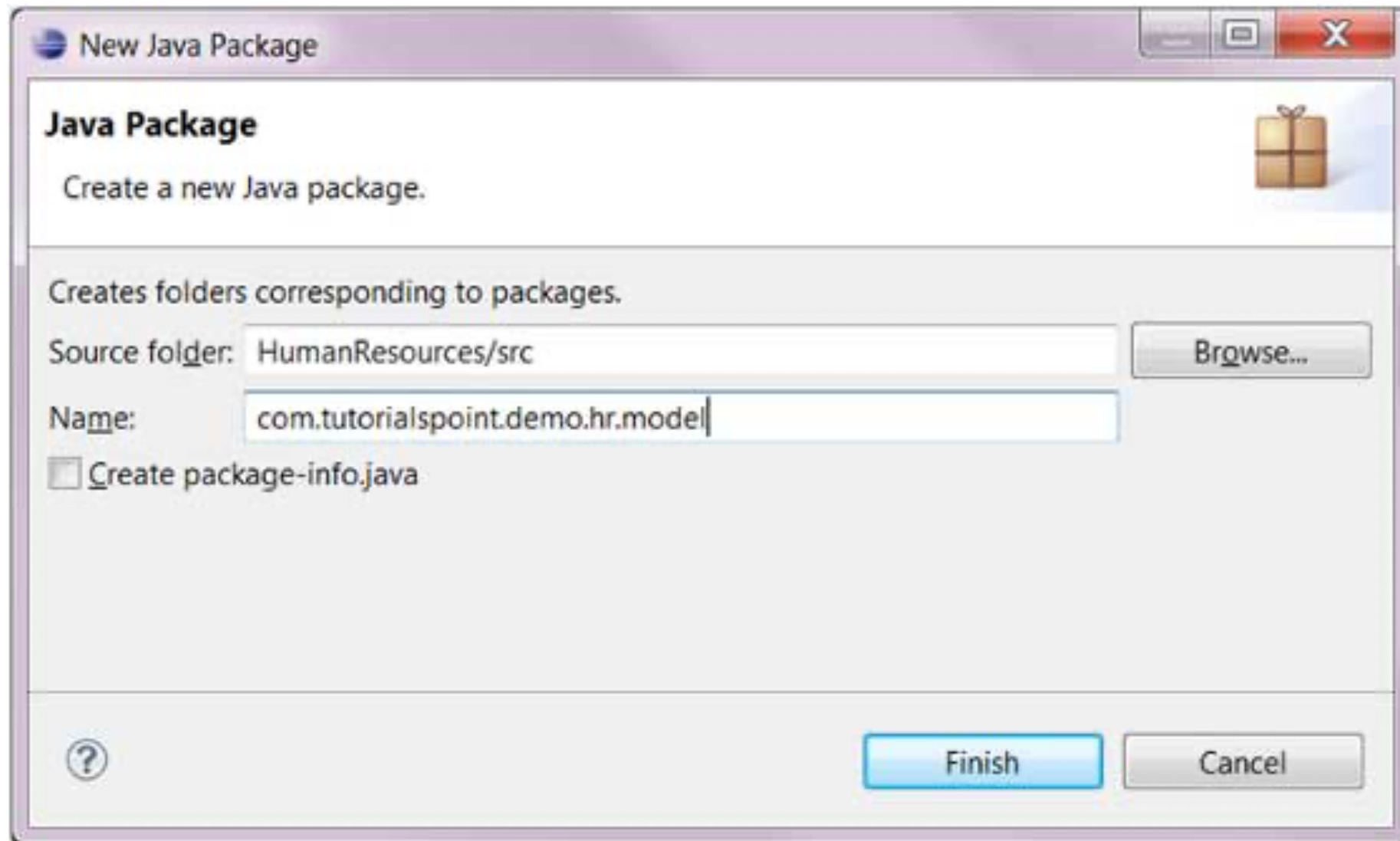


The Folder Wizard (File → New → Folder) can be used to create a new folder.



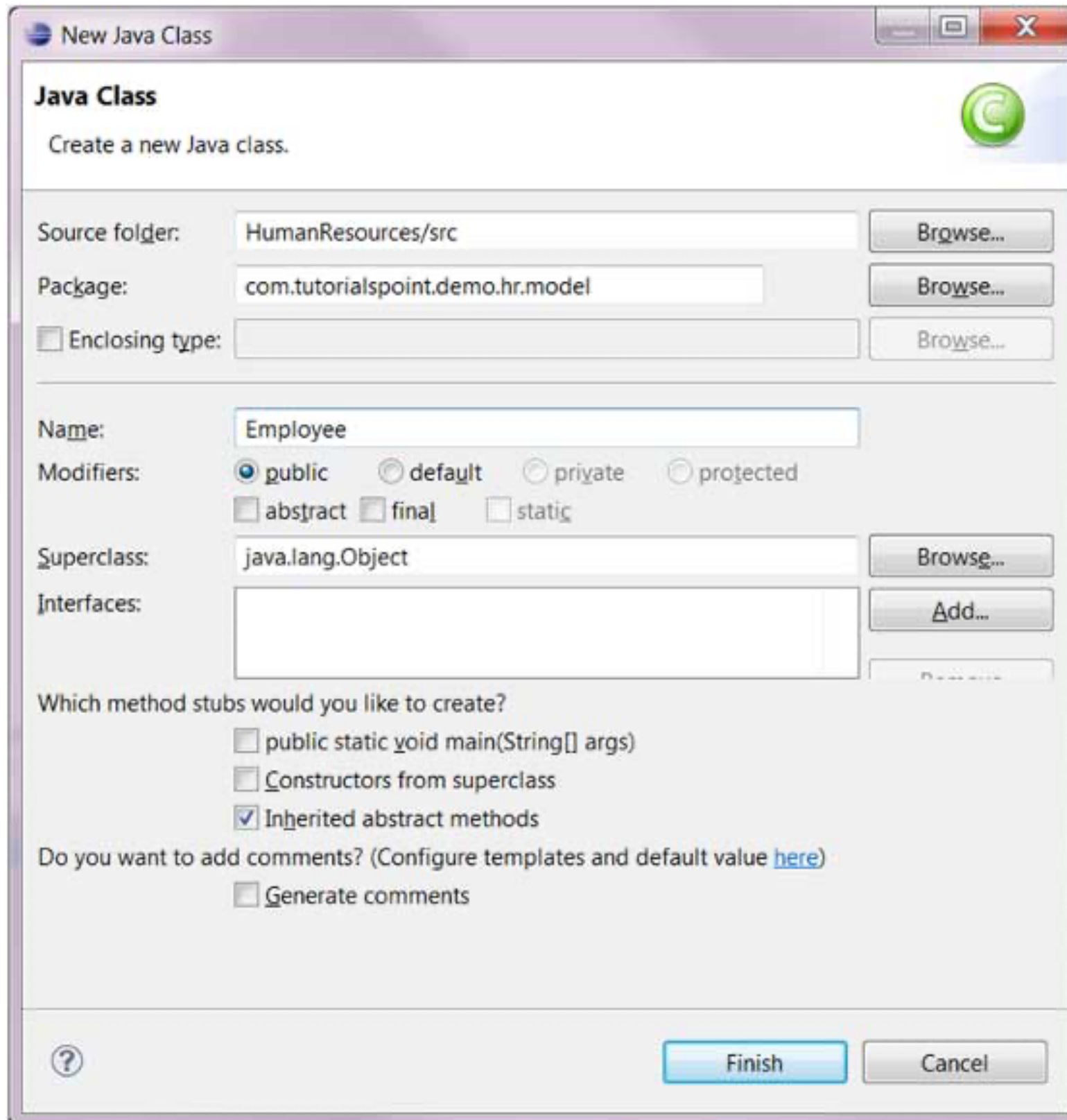
Creating New Package:

- By clicking on the File menu and selecting New → Package.
- By right click in the package explorer and selecting New → Package.
- By clicking on the package icon which is in the tool bar().



New Java Class Wizard

- By clicking on the File menu and selecting New → Class.
- By right clicking in the package explorer and selecting New → Class.
- By clicking on the class drop down button () and selecting class ().



New Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

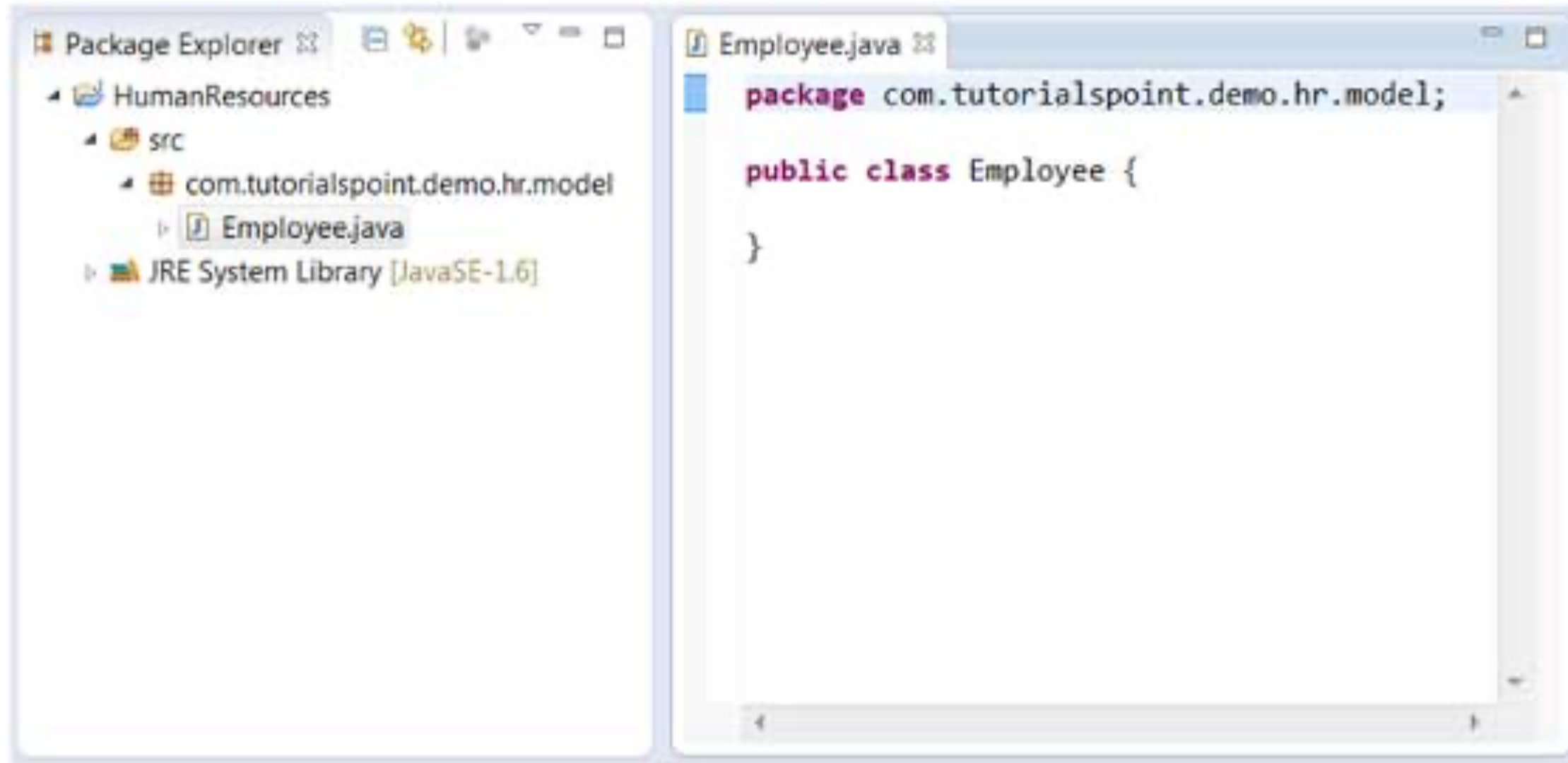
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Viewing the Newly Created Java class

The newly created class should appear in the Package Explorer view and a java editor instance that allows you to modify the new class.



Basic Syntax:

- **Case Sensitivity** - Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names** - For all class names, the first letter should be in Upper Case.
- **Method Names** - All method names should start with a Lower Case letter
- **Program File Name** - Name of the program file should exactly match the class name.
- **public static void main(String args[])** - The **main** method **must** be **public** so it can be found by the JVM when the class is loaded. Similarly, it **must** be **static** so that it can be called after loading the class, without **having** to create an instance of it. All methods **must have** a return type, which in this case is **void**. **String[] args** in Java is an array of **strings** which stores **arguments** passed by command line while starting a program. All the command line **arguments** are stored in that array.

Java Identifiers:

All Java components require names. Names used for classes, variables and methods are called identifiers

- All identifiers should begin with a letter (A to Z or a to z), currency character (\$) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- A keyword cannot be used as an identifier.
- Most importantly identifiers are case sensitive.
- Examples of legal identifiers: age, salary, _value
- Examples of illegal identifiers: 123abc, -salary

Java Modifiers:

it is possible to modify classes, methods, etc., by using modifiers.

- **Access Modifiers:** default, public, protected, private
- **Non-access Modifiers:** final, abstract, strictfp

Java Variables:

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non-static variables)

Java Keywords:

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

Basic Data Types

- **Primitive Data Types**
- **Reference/Object Data Types**

Primitive Data Types:

- **byte - 8 bit integer**
- **short - 16 bit integer**
- **int - 32 bit integer**
- **long - 64 bit integer**
- **float - single precision 32 bit floating point**
- **double - 64 bit floating point**
- **boolean - one bit of information**
- **char - 16 bit unicode character**

Reference Data Types:

- **Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy, etc.**
- **Class objects and various types of array variables come under reference data type.**
- **Default value of any reference variable is null.**
- **A reference variable can be used to refer to any object of the declared type or any compatible type.**
- **Example: `Animal animal = new Animal("giraffe");`**

Java Literals:

A literal is a source code representation of a fixed value.

```
byte a = 68; char a = 'A'
```

```
int decimal = 100; int octal  
= 0144; int hexa = 0x64;
```

Examples of string literals are:

```
"Hello World" "two\nlines"
```

```
"\"This is in quotes\""
```

Java language supports few special escape sequences for String and char literals as well.

Notation Character represented

`\n` Newline (0x0a)

`\r` Carriage return (0x0d)

`\f` Formfeed (0x0c)

`\b` Backspace (0x08)

`\s` Space (0x20)

`\t` Tab

`\"` Double quote

`\'` Single quote

`\\` Backslash

`\ddd` Octal character (ddd)

`\uxxxx` Hexadecimal UNICODE character (xxxx)

String class:

The String class represents character strings. All string literals in Java programs, such as "abc" , are implemented as instances of this class. Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings.

A Java String contains an immutable sequence of Unicode characters. Unlike C/ C++, where string is simply an array of char , A Java String is an object of the class java.lang You can assign a string literal directly into a String variable, instead of calling the constructor to create a String instance.

String class methods

- **public char charAt(int index) ...**
- **public String concat(String s) ...**
- **public boolean equalsIgnoreCase(String s) ...**
- **public int length() ...**
- **public String toUpperCase()**

String Buffer

StringBuffer in java is used to create modifiable String objects. This means that we can use StringBuffer to append, reverse, replace, concatenate and manipulate Strings or sequence of characters. StringBuffer and StringBuilder are called mutable because whenever we perform a modification on their objects their state gets changed. And they were created as mutable because of the immutable nature of String class.

String Builder

The StringBuilder Class. StringBuilder objects are like String objects, except that they can be modified. Internally, these objects are treated like variable-length arrays that contain a sequence of characters.

String is immutable whereas StringBuffer and StringBuider are mutable classes. StringBuffer is thread safe and synchronized whereas StringBuilder is not, thats why StringBuilder is more faster than StringBuffer.

Variable Types

variable provides us with named storage that our programs can manipulate.

```
int a, b, c;    // Declares three ints, a, b, and c.
int a = 10, b = 10; // Example of initialization
byte B = 22;    // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a';
```

Local variables:

- Local variables are declared in methods, constructors, or blocks.
- There is no default value for local variables
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor or block.

```
public class Test{  
    public void local_var()  
    {  
        int a = 0;  
        a = a + 7;  
        System.out.println("Variable a value is : " + a);  
    }  
  
    public static void main(String args[])  
    { Test test = new Test();  
      test.local_var();  
    }  
}
```

Instance variables:

- Instance variables are declared in a class, but outside a method, constructor or any block.
- Instance variable are created when an object is created with the use of 'new' keyword and destroys when the object is destroyed
- Instance variables can be declared in class level before or after use.
- The instance variables are visible for all methods, constructors and block in the class.
- Instance variables have default values
- Instance variable can be accessed directly by calling the variable name inside the class.

```
//import java.io.*;
```

```
public class Employee{  
    // this instance variable is visible for any child class.  
    public String name;  
  
    // salary variable is visible in Employee class only.  
    private double salary;  
  
    // The name variable is assigned in the constructor.  
    public Employee (String empName)  
    {  
        name = empName;  
    }  
}
```

```
// The salary variable is assigned a value.
public void setSalary(double empSal) {
    salary = empSal;
}

// This method prints the employee details.
public void printEmp() {
    System.out.println("name : " + name );
    System.out.println("salary :" + salary);
}

public static void main(String args[]) {
    Employee empOne = new Employee("Ransika");
    empOne.setSalary(1000);
    empOne.printEmp();
}
}
```

Class/static variables:

- **Class variable is also known as static variable are declared with the static keyword in the class**
- **Static variables are created when the program starts and destroyed when the program stops.**
- **Static variable are rarely declared as constant. Constants are variables that are declared as private / public , final and static. Constant variable never change from their initial values.**
- **Visibilities is similar to instance variables. However most static variables are declared public since they must be available for users of the class.**
- **Static variables can be accessed by calling with the class name.
ClassName.VariableName**
- **tinyurl.com/rlpfeedback**
- **Bhuvaneswari**

Advantages of static variable

It makes your program **memory efficient** (i.e., it saves memory).

```
public class Employee{  
    // salary variable is a private static variable  
    private static double salary;  
  
    // DEPARTMENT is a constant  
    public static final String DEPARTMENT = "Development ";  
  
    public static void main(String args[]){  
        salary = 1000;  
        System.out.println(DEPARTMENT+"average salary:"+salary) ;  
    }  
}
```

//Java Program to demonstrate the use of static variable

```
class Student{
    int rollno;//instance variable
    String name;
    static String college ="ITS";//static variable
    //constructor
    Student(int r, String n){
        rollno = r;
        name = n;
    }
    //method to display the values
    void display (){ System.out.println(rollno+" "+name+" "+college);}
}

//Test class to show the values of objects
public class TestStaticVariable1{
    public static void main(String args[]){
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan");

        s1.display();
        s2.display();
    }
}
```

Output:

```
111 Karan ITS
222 Aryan ITS
```

Access Modifiers

- **Visible to the package, the default. No modifiers are needed.**
- **Visible to the class only (private).**
- **Visible to the world (public).**
- **Visible to the package and all subclasses (protected)**

Default Access Modifier -- No keyword:

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc

```
String version ="1.5.1";
```

```
boolean processOrder(){ return true;
```

Private Access Modifier -- private:

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself. Private access modifier is the most restrictive access level. Class and interfaces cannot be private

```
public class Logger{ private String format;  
public String getFormat(){  
return this.format;  
}  
public void setFormat(String format){  
this.format = format;  
}
```

1) Private

```
class A{  
private int data=40;  
private void msg(){  
System.out.println("Hello java");}  
}
```

```
//public class Simple{  
public static void main(String args[]){  
    A obj=new A();  
    System.out.println(obj.data);//Compile Time Error  
    obj.msg();//Compile Time Error  
}  
//}
```

A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

2) Default

//save by A.java

```
package pack;  
class A{  
    void msg(){  
System.out.println("Hello");  
}  
}
```

```
//save by B.java
package mypack;
import pack.*;
class B{
    public static void main(String args[]){
        A obj = new A();//Compile Time Error
        obj.msg();//Compile Time Error
    }
}
```

We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

3) Protected

- The protected access modifier is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
- It provides more accessibility than the default modifier.

```
//save by A.java
package pack;
public class A{
    protected void msg(){
        System.out.println("Hello");}
}
```

//save by B.java

```
package mypack;  
import pack.*;
```

```
class B extends A{  
    public static void main(String args[]){  
        B obj = new B();  
        obj.msg();  
    }  
}
```

Output:Hello

4) Public

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

//save by A.java

```
package pack;  
public class A{  
    public void msg(){  
        System.out.println("Hello");  
    }  
}
```

//save by B.java

```
package mypack;  
import pack.*;  
  
class B{  
    public static void main(String args[]){  
        A obj = new A();  
        obj.msg();  
    }  
}
```

Basic Operators

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

```

public class Test{

public static void main(String args[]){
int a =10;
int b =20;
int c =25;
int d =25;

System.out.println("a + b = "+(a + b));
System.out.println("a - b = "+(a - b));
System.out.println("a * b = "+(a * b));
System.out.println("b / a = "+(b / a));
System.out.println("b % a = "+(b % a));
System.out.println("c % a = "+(c % a));
System.out.println("a++= "+(a++)); // 1 0 + +
System.out.println("a-- = "+(a--));//11 --
// Check the difference in d++ and ++d
System.out.println("d++= "+(d++)); // p o s t i n c r e m e n t
System.out.println("++d= "+(++d));//pre increment
}

```

Output:

Output:

```

a + b =30
a - b =-10
a * b =200
b / a =2
b % a =0
c % a =5
a++=10
a--=11
d++=25
++d =27

```


Relational Operators:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of right operand is greater than or equal to the value	(A >= B) is not true.
<=	Checks if the value of left operand is greater than or equal to the value	(A <= B) is true.

```
public class Test{  
  
    public static void main(String args[]){  
        int a =10;  
        int b =20;  
        System.out.println("a == b = "+(a == b));  
        System.out.println("a != b = "+(a != b));  
        System.out.println("a > b = "+(a > b));  
        System.out.println("a < b = "+(a < b));  
        System.out.println("b >= a = "+(b >= a));  
        System.out.println("b <= a = "+(b <= a));  
    }  
}
```

Output:

a == b =false

a != b =true

a > b =false

a < b =true

b >= a =true

b <= a =false

The Bitwise Operators:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
 	Binary OR Operator copies a bit if it exists in either operand.	(A B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

```

public class Test{

public static void main(String args[]){
int a =60;  /* 60 = 0011 1100 */
int b =13;  /* 13 = 0000 1101 */
int c =0;


    c = a & b; /* 12 = 0000 1100 */
    System.out.println("a & b = "+ c );

    c = a | b; /* 61 = 0011 1101 */
    System.out.println("a | b = "+ c );

    c = a ^ b; /* 49 = 0011 0001 */
    System.out.println("a ^ b = "+ c );

    c = ~a; /* -61 = 1100 0011 */
    System.out.println("~a = "+ c );

    c = a <<2; /* 240 = 1111 0000 */
    System.out.println("a << 2 = "+ c );

    c = a >>2; /* 215 = 1111 */
    System.out.println("a >> 2 = "+ c );

    c = a >>>2; /* 215 = 0000 1111 */
    System.out.println("a >>> 2 = "+ c );
}

```

Output:

```

a & b =12
a | b =61
a ^ b =49
~a =-61
a <<2=240
a >>15
a >>>15

```

Logical Operators:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

```
public class Test{

    public static void main(String args[]){
        boolean a =true;
        boolean b =false;
        System.out.println("a && b = "+(a&&b));
        System.out.println("a || b = "+(a||b));
        System.out.println("!(a && b) = "+!(a && b));
    }
}
```

Output:

```
a && b =false
a || b =true
!(a && b)=true
```

Assignment Operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C=A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
 =	bitwise inclusive OR and assignment operator	C = 2 is same as C = C 2

Increment and Decrement Operators

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

```
        // declare variables  
        int a = 12, b = 12;  
        int result1, result2;
```

```
        // original value  
        System.out.println("Value of a: " + a);
```

```
        // increment operator  
        result1 = ++a;  
        System.out.println("After increment: " + result1);
```

```
        System.out.println("Value of b: " + b);
```

```
        // decrement operator  
        result2 = --b;  
        System.out.println("After decrement: " + result2);  
    }  
}
```

Output:

Value of a: 12

After increment: 13

Value of b: 12

After decrement: 11

Misc Operators

Conditional Operator (?:):

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions.

variable x = (expression) ? value if true : value if false

```
public class Test{  
public static void main(String args[]){  
int a , b;  
a =10;  
b =(a ==1) ?20:30;  
System.out.println("Value of b is : "+ b );  
b =(a ==10) ?20:30;  
System.out.println("Value of b is : "+ b );  
}  
}
```

Output:

Value of b is:30

Value of b is:20

instanceof Operator:

This operator is used only for object reference variables.

```
(Object reference variable ) instanceof (class/interface type)
```

```
class Simple1{  
    public static void main(String args[]){  
        Simple1 s=new Simple1();  
        System.out.println(s instanceof Simple1);//true  
    }  
}
```

Output:

true

Precedence of Java Operators:

Operator precedence determines the grouping of terms in an expression.

For example, $x = 7 + 3 * 2$; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with $3*2$ and then adds into 7

BODMAS - Bracket Order Division Multiplication Addition Subtraction

Bracket = ()

Order = Square root , Power of

Division = /

Multiplication = *

Addition = +

Subtraction = -

Loop Control

- while Loop
- do...while Loop
- for Loop

while Loop:

```
public class Test{
public static void main(String args[]) {
int x =10;
while( x <20) {
System.out.println("value of x : "+ x );
x++;
//System.out.print("\n");
}
}
}
```

Output:

```
value of x :10
value of x :11
value of x :12
value of x :13
value of x :14
value of x :15
value of x :16
value of x :17
value of x :18
Value of x : 19
```

The do...while Loop:

Syntax:

```
do
{
//Statements
}while(Boolean_expression);
```

```
public class Test{
public static void main(String args[]){
int x =20;
do{
System.out.print("value of x : "+ x );
x++;
System.out.print("\n");
}while( x <20);
}
}
```

Output:

```
value of x :10
value of x :11
value of x :12
```

The for Loop:

Syntax:

```
for(initialization; Boolean_expression; update)
{
//Statements
}
```

```
public class Test{
```

```
public static void main(String args[]) {
```

```
for(int x =10; x <20; x = x+1) {
```

```
System.out.print("value of x : "+ x );
```

```
System.out.print("\n");
```

```
}
```

```
}
```

```
}
```

Output:

value of x :10

value of x :11

value of x :12

Enhanced for loop in Java:

Syntax:

```
for (declaration : expression)
{
//Statements
}
```

- **Declaration:** The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.
- **Expression:** This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

```
public class Test{

public static void main(String args[]){
int[] numbers ={10,20,30,40,50};

for(int x : numbers ){
System.out.print(x);
System.out.print(",");
}
System.out.print("\n");
String[] names={"James","Larry","Tom","Lacy"};
for(String name : names ){
System.out.print( name );
System.out.print(",");
}
}
}
```

Output:

10,20,30,40,50,
James,Larry,Tom,Lacy,

The break Keyword:

```
public class Test{  
  
    public static void main (String  
    args[]) {  
        int[] numbers = {10, 20, 30, 40, 50};  
        for (int x : numbers) {  
            if (x == 30) {  
                break;  
            }  
            System.out.print ( x );  
            System.out.print ("\n");  
        }  
    }  
}
```

Output:

10
20

The continue Keyword:

```
public class Test{

public static void main(String args[]) {
int[] numbers ={10,20,30,40,50};

for(int x : numbers){
if( x ==30){
    continue;
}
System.out.print( x );
System.out.print("\n");
}
}
}
```

Output:

10
20
40
50

Decision Making

The if Statement:

An if statement consists of a Boolean expression followed by one or more statements.

Syntax:

```
if (Boolean_expression)
{
//Statements will execute if the Boolean expression is true
}
```

```
public class Test{
```

```
public static void main(String args[]) {
int x =10;
```

```
if( x <20) {
System.out.print("This is if statement");
}
}
}
```

Output:

This is if statement

The if...else Statement:

An if statement can be followed by an optional e/se statement, which executes when the Boolean expression is false.

Syntax:

```
if(Boolean_expression){  
    //Executes when the Boolean expression is true  
}  
else{  
    //Executes when the Boolean expression is false  
}
```

```
public class Test{  
  
    public static void main(String args[]){  
        int x =30;  
  
        if(x <20){  
            System.out.print("This is if statement");  
        }else{  
            System.out.print("This is else statement");  
        }  
    }  
}
```

Output:

This is else statement

The if...else if...else Statement:

Syntax:

```
if(Boolean_expression1){  
    //Executes when the Boolean expression 1 is true  
}  
else if(Boolean_expression2){  
    //Executes when the Boolean expression 2 is true  
}  
else if(Boolean_expression3){  
    //Executes when the Boolean expression 3 is true  
}  
else{  
    //Executes when the none of the above condition is true.  
}
```

```
public class Test{
```

```
    public static void main(String args[]){
```

```
        int x =30;
```

```
        if( x ==10){
```

```
            System.out.print("Value of X is 10");
```

```
        }else if( x ==20){
```

```
            System.out.print("Value of X is 20");
```

```
        }else if( x ==30){
```

```
            System.out.print("Value of X is 30");
```

```
        }else{
```

```
            System.out.print("This is else statement");
```

```
        }
```

```
    }
```

```
}
```

Output:

Value of X is30

Nested if...else Statement:

Syntax:

```
if(Boolean_expression1){  
    //Executes when the Boolean expression 1 is true  
    if(Boolean_expression2){  
        //Executes when the Boolean expression 2 is true  
    }  
}
```

Output:

X =30and Y =10

```
public class Test{  
  
    public static void main(String args[]){ int x =30;  
    int y =10;  
  
    if( x ==30){  
        if( y ==10){  
            System.out.print("X = 30 and Y = 10");  
        }  
    }  
}
```

The switch Statement:

```
switch(expression){  
    case value :  
        //Statements  
        break;//optional  
    case value :  
        //Statements  
        break;//optional  
    //You can have any number of case statements.  
    default://Optional  
        //Statements  
}
```

```

import java.util.*;
public class Test{
public static void main(String args[]){
Scanner scan=new Scanner(System.in);
System.out.println("Enter the grade:");
char ch=scan.next().charAt(0);
switch(ch)
{
case 'A': System.out.println("Excellent!");
break;
case 'B':
case 'C': System.out.println("Well done");
break;
case 'D':
System.out.println("You passed");
case 'F':
System.out.println("Better try again");
break;

default:
System.out.println("Invalid grade");
}
System.out.println("Your grade is "+ ch);
}
}

```

Output:

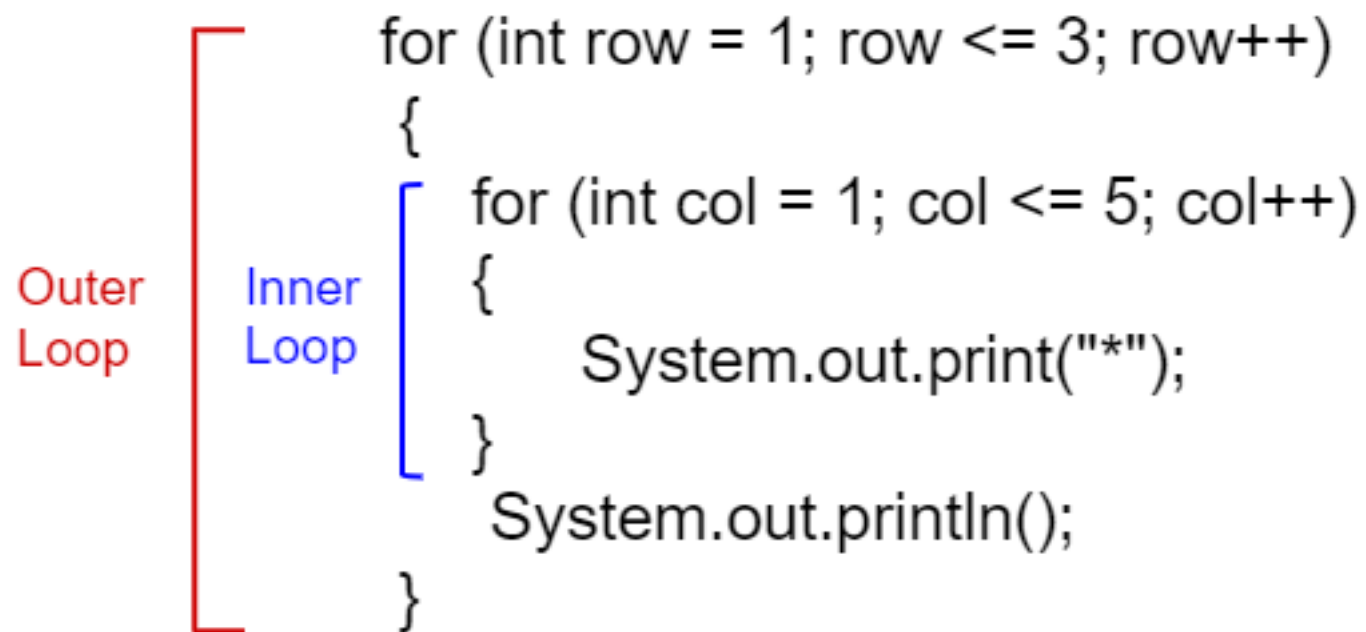
```

$ java Test a Invalid grade Your grade is a
$ java Test A Excellent!
Your grade is a A
$ java Test C Welldone
Your grade is a C
$

```

Nested Loops

A *nested loop* is a (inner) loop that appears in the loop body of another (outer) loop. The inner or outer loop can be any type: **while**, **do while**, or **for**. For example, the inner loop can be a **while** loop while an outer loop can be a **for** loop.



The diagram illustrates nested loops using a code example. A red bracket on the left, labeled "Outer Loop", spans the entire code block. A blue bracket on the left, labeled "Inner Loop", spans the inner loop's body. The code is as follows:

```
for (int row = 1; row <= 3; row++)  
{  
    for (int col = 1; col <= 5; col++)  
    {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

Labelled and Unlabelled Statements:

The *continue* Statement, Without a Label

```
public class Unlabelledstmt{

public static void main(String args[]){
int[] numbers ={10,20,30,40,50};

for(int x : numbers){
if( x ==30){
    continue;
}
System.out.print( x );
System.out.print("\n");
}
}
}
```

```
public class LabelledStmt{

public static void main(String args[]){
int[] numbers ={10,20,30,40,50};

start: for(int x : numbers){
if( x ==30){
    continue start;
}
System.out.print( x );
System.out.print("\n");
}
}
}
```

```

public class SimpleContinueDemo {

    public static void main(String[] args) {
        String[] listOfNames = { "Ravi", "Soma",
            "null", "Colin", "Harry", "null",
            "Smith" };

        for (int i = 0; i < listOfNames.length; i++) {
            if (listOfNames[i].equals("null"))
                continue;
            System.out.println(listOfNames[i]);
        }
    }
}

```

The *continue* Statement, with a Label

```

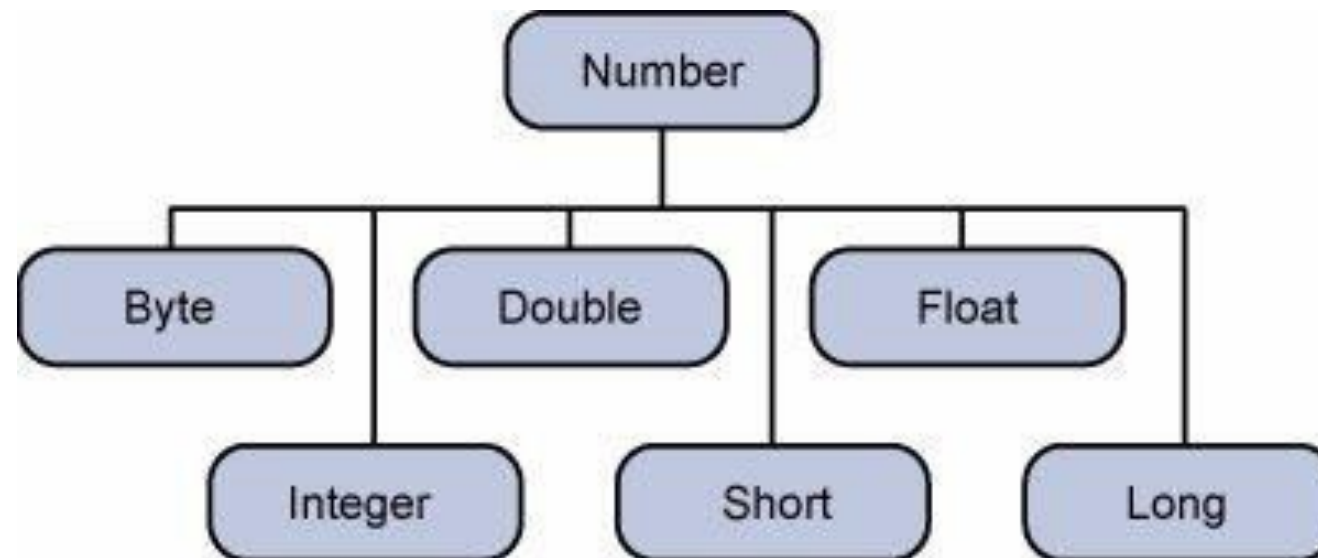
public class LabeledContinueDemo {

    public static void main(String[] args) {

        start: for (int i = 0; i < 5; i++) {
            System.out.println();
            for (int j = 0; j < 10; j++) {
                System.out.print("#");
                if (j >= i)
                    continue start;
            }
            System.out.println("This will never"
                + " be printed");
        }
    }
}

```


Numbers



```
public class Test{  
  
    public static void main(String args[]){  
        Integer x =5; // boxes int to an Integer object  
        x =x +10; // unboxes the Integer to a int  
        System.out.println(x);  
    }  
}
```

Number Methods:

SN	Methods with Description
1	xxxValue() Converts the value of <i>this</i> Number object to the xxx data type and returned it.
2	compareTo() Compares <i>this</i> Number object to the argument.
3	equals() Determines whether <i>this</i> number object is equal to the argument.
4	valueOf() Returns an Integer object holding the value of the specified primitive.
5	toString() Returns a String object representing the value of specified int or Integer.
6	parseInt() This method is used to get the primitive data type of a certain String.
7	abs() Returns the absolute value of the argument.
8	ceil() Returns the smallest integer that is greater than or equal to the argument. Returned as a double.

SN	Methods with Description
9	floor() Returns the largest integer that is less than or equal to the argument. Returned as a double.
10	rint() Returns the integer that is closest in value to the argument. Returned as a double.
11	round() Returns the closest long or int, as indicated by the method's return type, to the argument.
12	min() Returns the smaller of the two arguments.
13	max() Returns the larger of the two arguments.
14	exp() Returns the base of the natural logarithms, e, to the power of the argument.
15	log() Returns the natural logarithm of the argument.
16	pow() Returns the value of the first argument raised to the power of the second argument.

17	<code>sqrt()</code> Returns the square root of the argument.
18	<code>sin()</code> Returns the sine of the specified double value.
19	<code>cos()</code> Returns the cosine of the specified double value.
20	<code>tan()</code> Returns the tangent of the specified double value.
21	<code>asin()</code> Returns the arcsine of the specified double value.
22	<code>acos()</code> Returns the arccosine of the specified double value.
23	<code>atan()</code> Returns the arctangent of the specified double value.
24	<code>atan2()</code> Converts rectangular coordinates (x, y) to polar coordinate (r, theta) and returns theta.
25	<code>toDegrees()</code> Converts the argument to degrees
26	<code>toRadians()</code> Converts the argument to radians.
27	<code>random()</code> Returns a random number.

Characters

```
char ch ='a';

// Unicode for uppercase Greek omega character char
uniChar =' \u039A';

// an array of chars
char[] charArray ={ 'a' , 'b' , 'c' , 'd' , 'e' };
```

Escape Sequence	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point.
\f	Inserts a form feed in the text at this point.
\'	Inserts a single quote character in the text at this point.
\"	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.

Character Methods:

SN	Methods with Description
1	isLetter() Determines whether the specified char value is a letter.
2	isDigit() Determines whether the specified char value is a digit.
3	isWhitespace() Determines whether the specified char value is white space.
4	isUpperCase() Determines whether the specified char value is uppercase.
5	isLowerCase() Determines whether the specified char value is lowercase.
6	toUpperCase() Returns the uppercase form of the specified char value.
7	toLowerCase() Returns the lowercase form of the specified char value.
8	toString() Returns a String object representing the specified character value that is, a one-character string.

```
public class Test{

public static void main(String args[])
{ System.out.println(Character.isLetter('c'));
System.out.println(Character.isLetter('5'));
}
}
```

```
public class Test{

public static void main(String args[])
{ System.out.println(Character.isDigit('c'));
System.out.println(Character.isDigit('5'));
}
}
```

```
public class Test{

public static void main(String args[])
{ System.out.println(Character.isLowerCase('c'));
System.out.println(Character.isLowerCase('C'));
System.out.println(Character.isLowerCase('\n'));
System.out.println(Character.isLowerCase('\t'));
}
}
```

Strings:

Creating strings:

```
public class StringDemo{  
  
    public static void main(String args[]){  
        String string1 ="saw I was ";  
        System.out.println("Dot "+ string1 +"Tod");  
    }  
}
```

Concatenating Strings:

```
"My name is ".concat("Zara");
```

Creating Format Strings:

```
String fs;  
fs =String.format("The value of the float variable  
is "+ "%f, while the value of the integer "+  
  
"variable is %d, and the string "+  
"is %s", floatVar, intVar, stringVar);  
System.out.println(fs);  
  
"%f, while the value of the integer "+  
"variable is %d, and the string "+  
"is %s", floatVar, intVar, stringVar);
```


String Methods:

S N	Methods with Description
1	<u>char charAt(int index)</u> Returns the character at the specified index.
2	<u>int compareTo(Object o)</u> Compares this String to another Object.
3	<u>int compareTo(String anotherString)</u> Compares two strings lexicographically.
4	<u>int compareToIgnoreCase(String str)</u> Compares two strings lexicographically, ignoring case differences.
5	<u>String concat(String str)</u> Concatenates the specified string to the end of this string.
6	<u>boolean contentEquals(StringBuffer sb)</u> Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	<u>static String copyValueOf(char[] data)</u> Returns a String that represents the character sequence in the array specified.

8 static String copyValueOf(char[] data, int offset, int count)

Returns a String that represents the character sequence in the array specified.

9 boolean endsWith(String suffix)

Tests if this string ends with the specified suffix.

10 boolean equals(Object anObject) Compares this string to the specified object.

11 boolean equalsIgnoreCase(String anotherString)

Compares this String to another String, ignoring case considerations.

12 byte getBytes()

Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

13 byte[] getBytes(String charsetName)

Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.

String class:

The String class represents character strings. All string literals in Java programs, such as "abc" , are implemented as instances of this class. Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings.

A Java String contains an immutable sequence of Unicode characters. Unlike C/ C++, where string is simply an array of char , A Java String is an object of the class java.lang You can assign a string literal directly into a String variable, instead of calling the constructor to create a String instance.

String class methods

- **public char charAt(int index) ...**
- **public String concat(String s) ...**
- **public boolean equalsIgnoreCase(String s) ...**
- **public int length() ...**
- **public String toUpperCase()**

String Buffer

StringBuffer in java is used to create modifiable String objects. This means that we can use StringBuffer to append, reverse, replace, concatenate and manipulate Strings or sequence of characters.

StringBuffer and **StringBuilder** are called mutable because whenever we perform a modification on their objects their state gets changed. And they were created as mutable because of the immutable nature of String class.

String Builder

The StringBuilder Class. **StringBuilder** objects are like String objects, except that they can be modified. Internally, these objects are treated like variable-length arrays that contain a sequence of characters.

String is immutable whereas **StringBuffer** and **StringBuider** are mutable classes. **StringBuffer** is thread safe and synchronized whereas **StringBuilder** is not, thats why **StringBuilder** is more faster than **StringBuffer**.

StringBuffer

VS

StringBuilder

StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.

StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.

StringBuffer is less efficient than StringBuilder.

StringBuilder is more efficient than StringBuffer.

```
public class StringCompareEmp{
    public static void main(String args[]){
        String str = "Hello World";
        String anotherString = "hello world";
        Object objStr = str;

        System.out.println( str.compareTo(anotherString) );
        System.out.println( str.compareToIgnoreCase(anotherString) );
        System.out.println( str.compareTo(objStr.toString()));
    }
}
```

```
public class StringCompareequl{
    public static void main(String []args){
        String s1 = "google";
        String s2 = "google";
        String s3 = new String ("google world");
        System.out.println(s1.equals(s2));
        System.out.println(s2.equals(s3));
    }
}
```

```
public class StringCompareequl{
    public static void main(String []args){
        String s1 = "google";
        String s2 = "google";
        String s3 = new String ("google world");
        System.out.println(s1 == s2);
        System.out.println(s2 == s3);
    }
}
```