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.

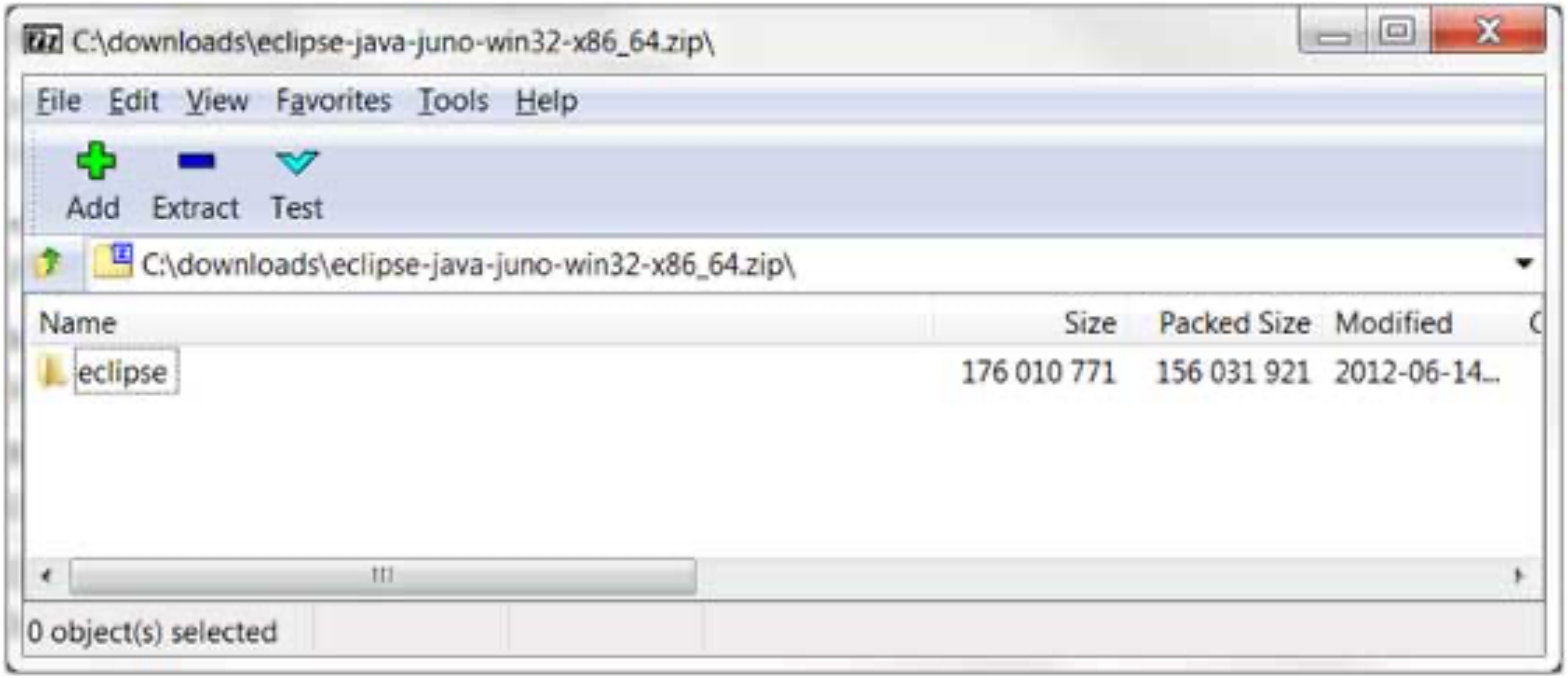
Eclipse:

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

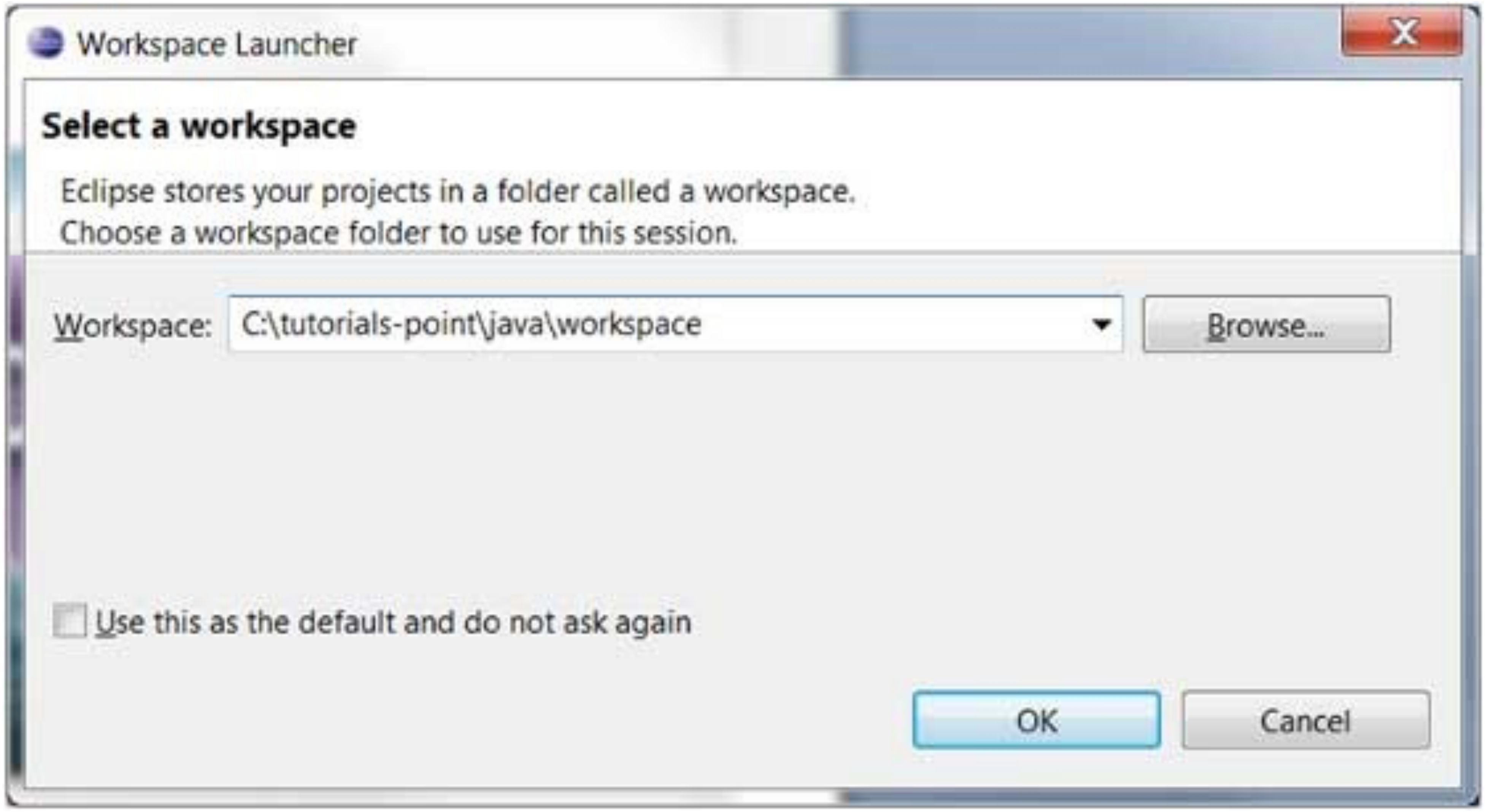


C:\downloads\eclipse-java-juno-win32-x86_64.zip\

Name	Size	Packed Size	Modified
eclipse	176 010 771	156 031 921	2012-06-14...

0 object(s) selected

Launching Eclipse:


Workspace Launcher

Select a workspace

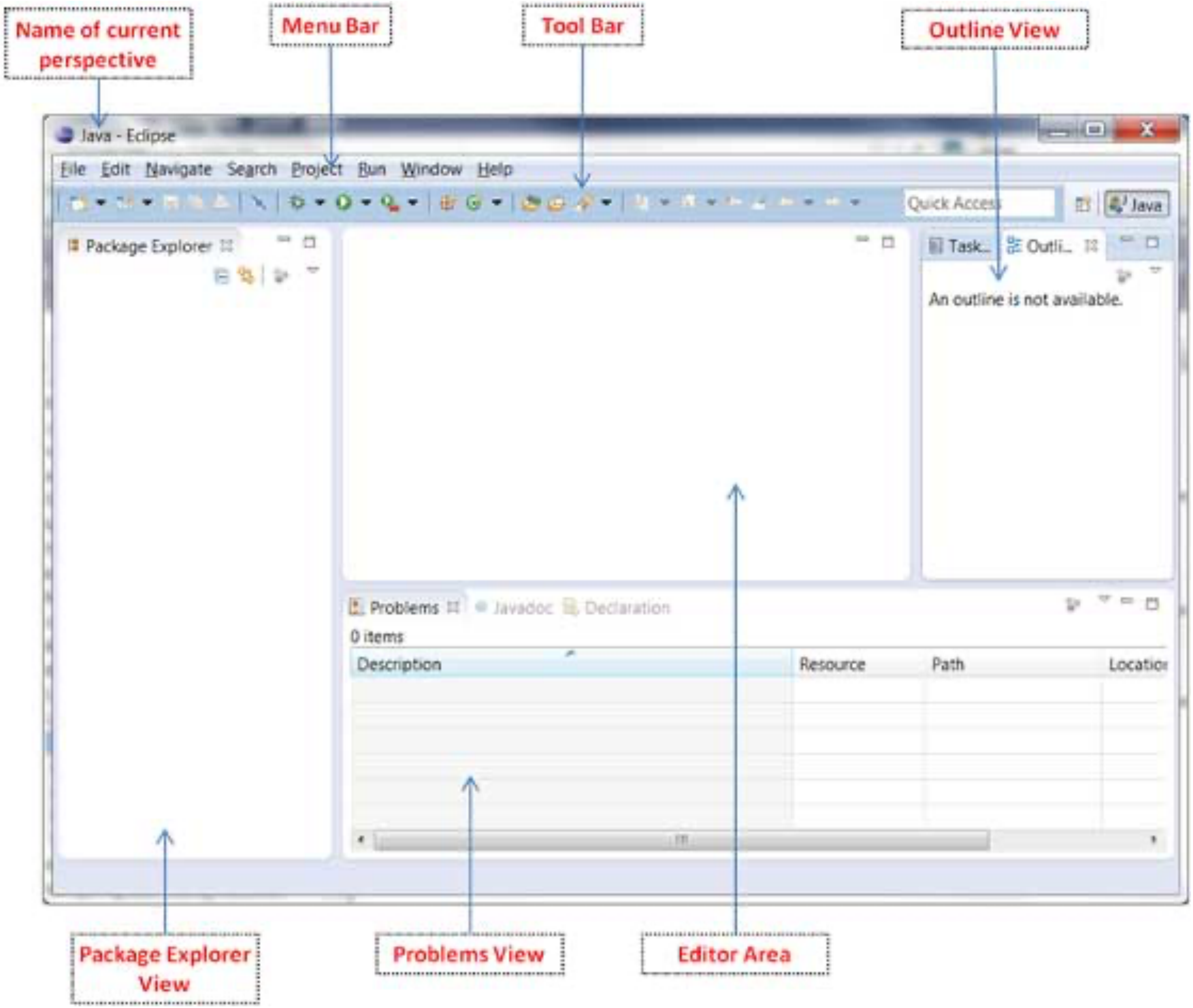
Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\tutorials-point\java\workspace

Parts of an Eclipse Window

The major visible parts of an eclipse window are −

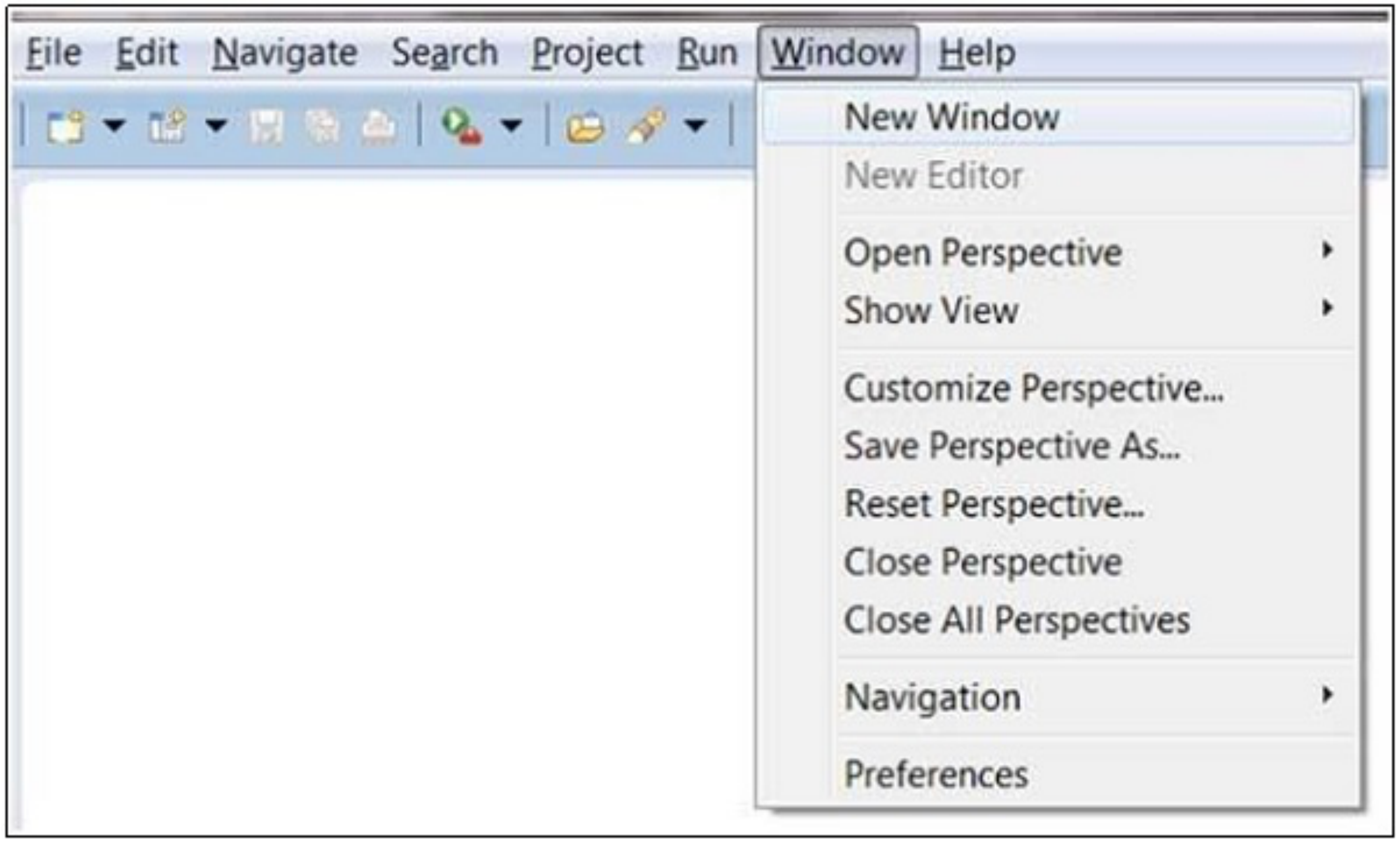
- Views
- Editors (all appear in one editor area)
- Menu Bar
- Toolbar


Name of current perspective
Menu Bar
Tool Bar
Outline View
Java - Eclipse
File Edit Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
Task...
Outli...
An outline is not available.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Package Explorer View
Problems View
Editor Area

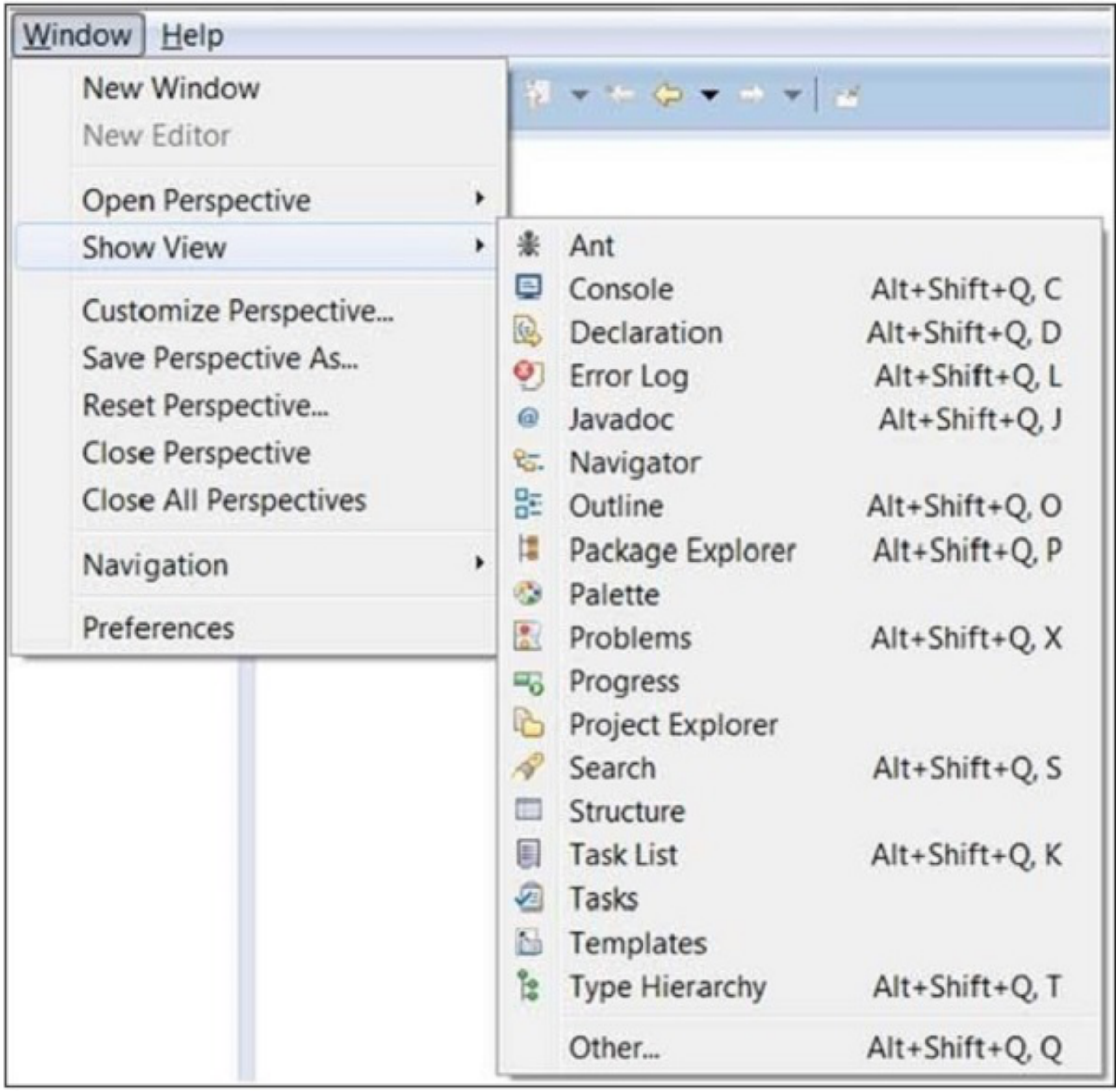
Typical Eclipse Menus

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

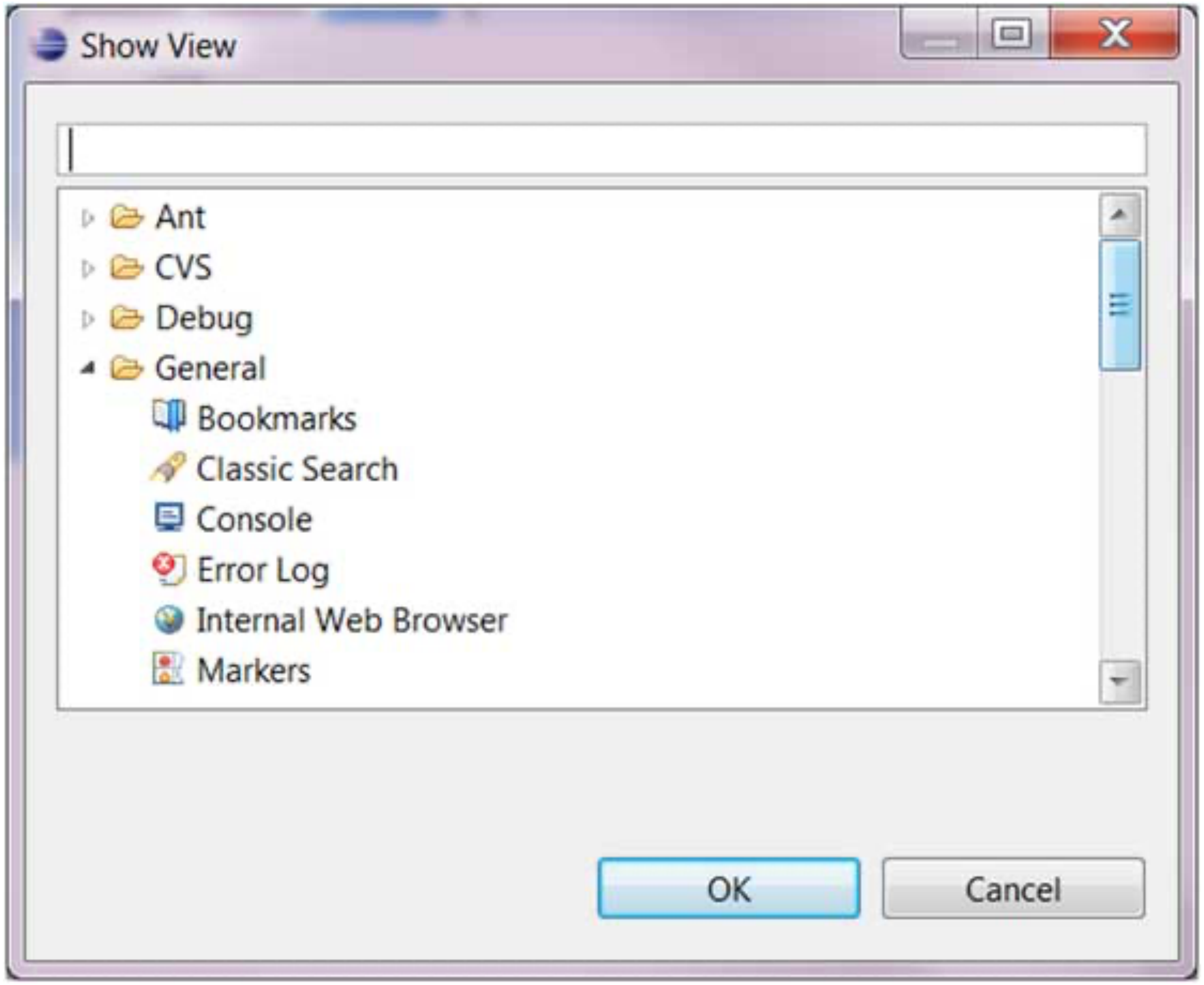
- File menu
- Edit menu
- Navigate menu
- Search menu
- Project menu
- Run menu
- Window menu
- Help menu



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.

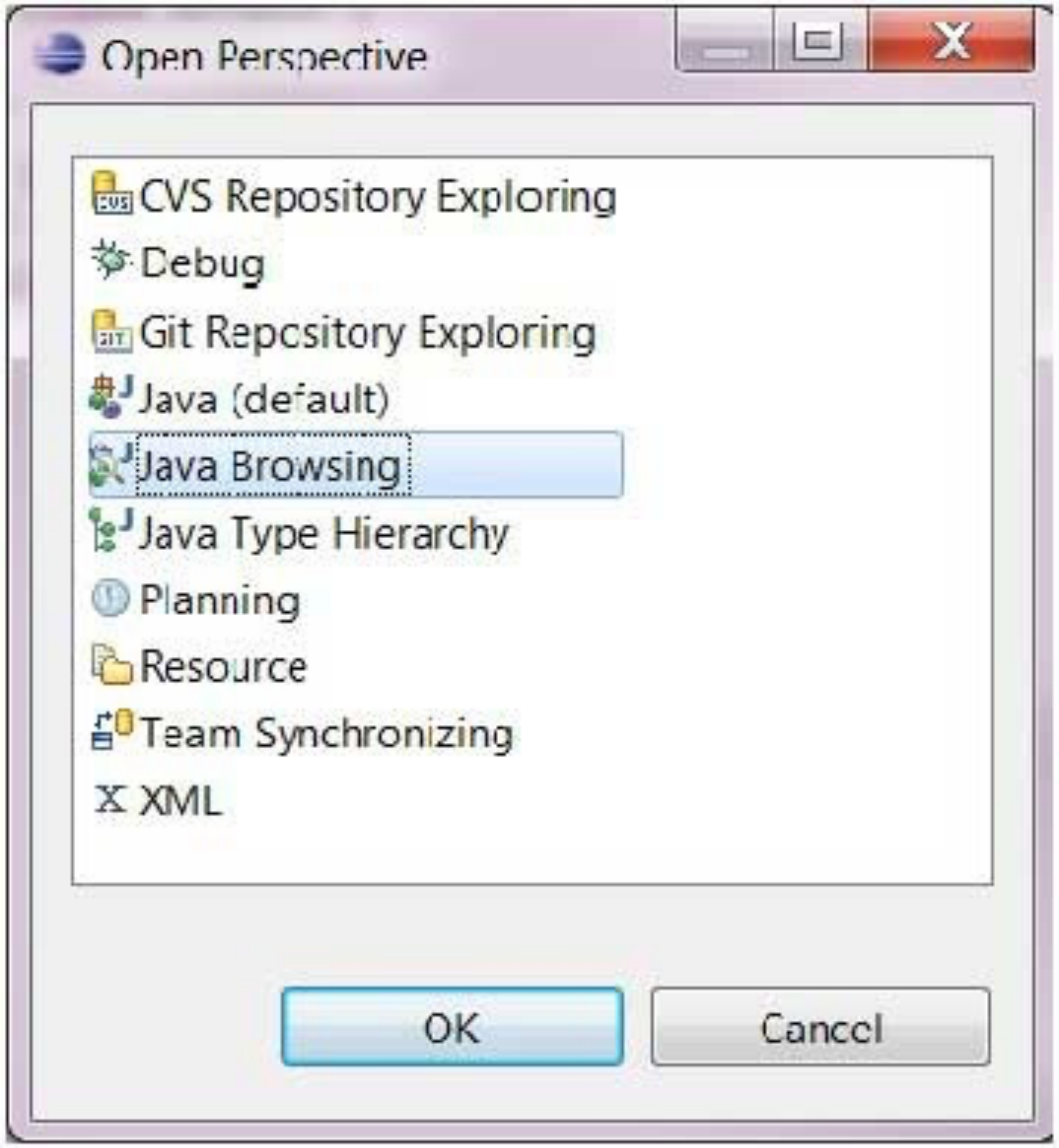


What is a Perspective?

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

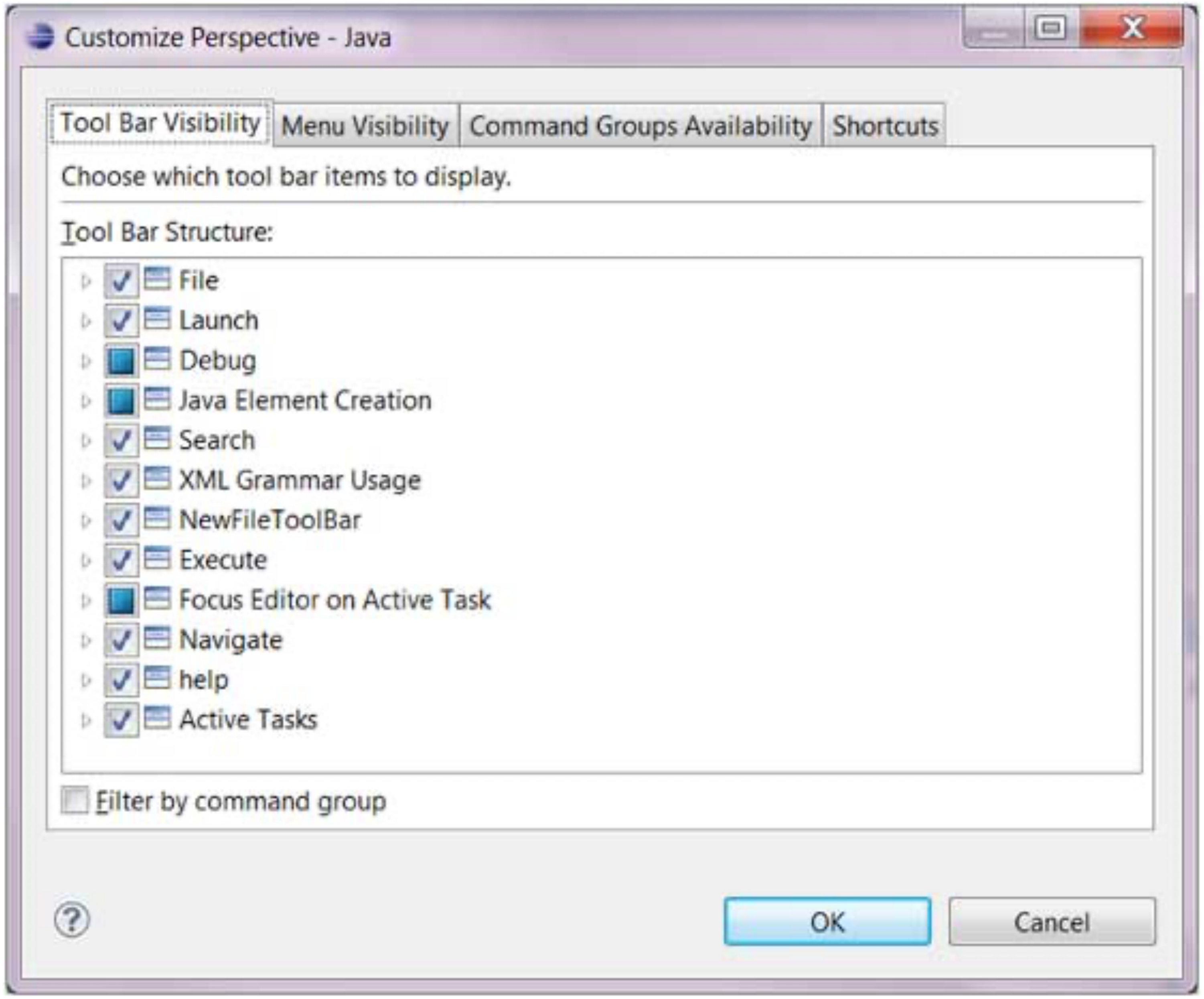

Open Perspective

- CVS Repository Exploring
- Debug
- Git Repository Exploring
- Java (default)
- Java Browsing
- Java Type Hierarchy
- Planning
- Resource
- Team Synchronizing
- XML

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.



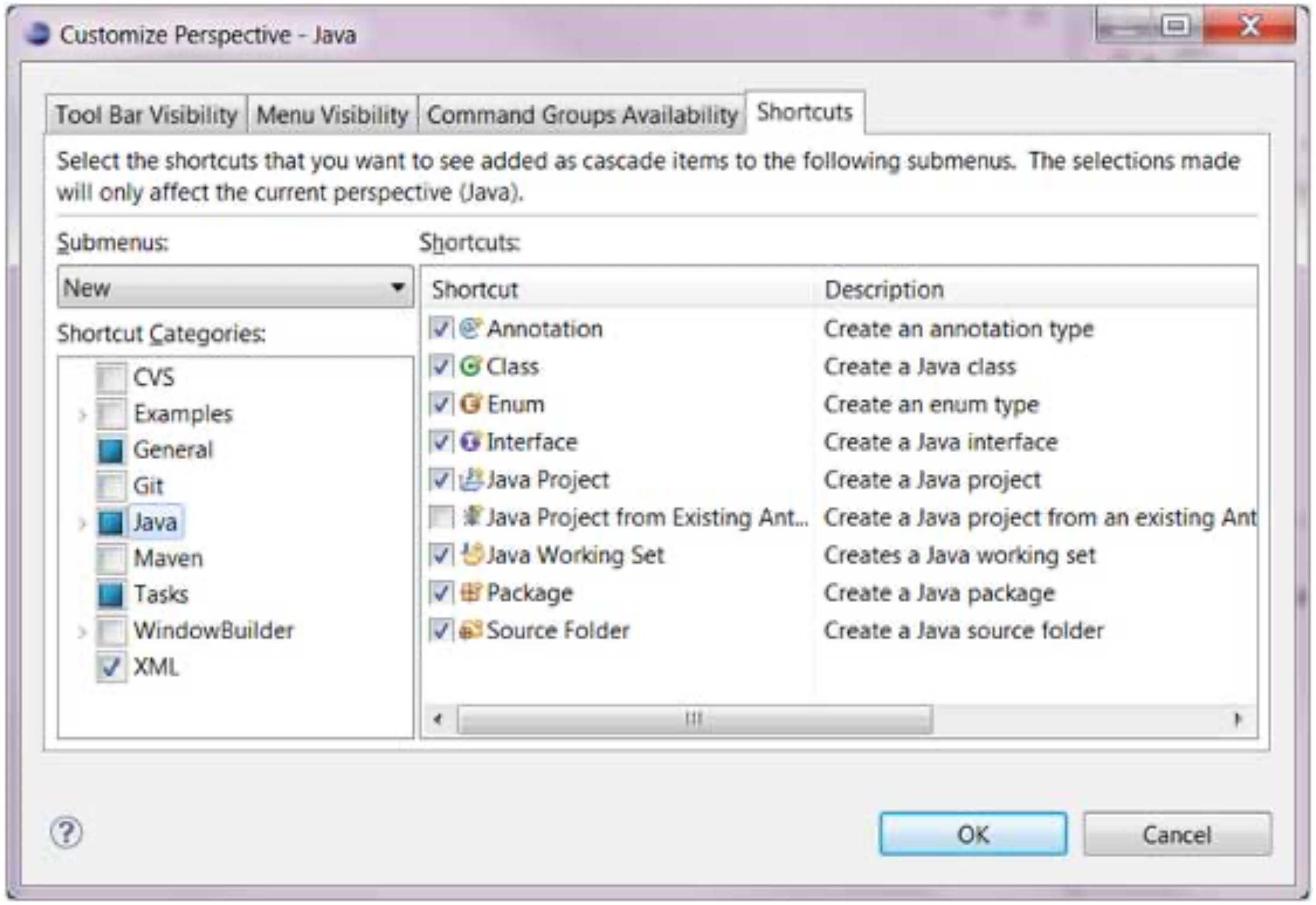
Choose which tool bar items to display.

Tool Bar Structure:

- ✓ File
- ✓ Launch
- ■ Debug
- ■ Java Element Creation
- ✓ Search
- ✓ XML Grammar Usage
- ✓ NewFileToolBar
- ✓ Execute
- ■ Focus Editor on Active Task
- ✓ Navigate
- ✓ help
- ✓ Active Tasks

☐ Filter by command group

- 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.



Select the shortcuts that you want to see added as cascade items to the following submenus. The selections made will only affect the current perspective (Java).

Submenus: New

Shortcut Categories:
- [ ] CVS
- [ ] Examples
- ■ General
- [ ] Git
- ■ Java
- [ ] Maven
- ■ Tasks
- [ ] WindowBuilder
- [x] XML

Shortcuts:

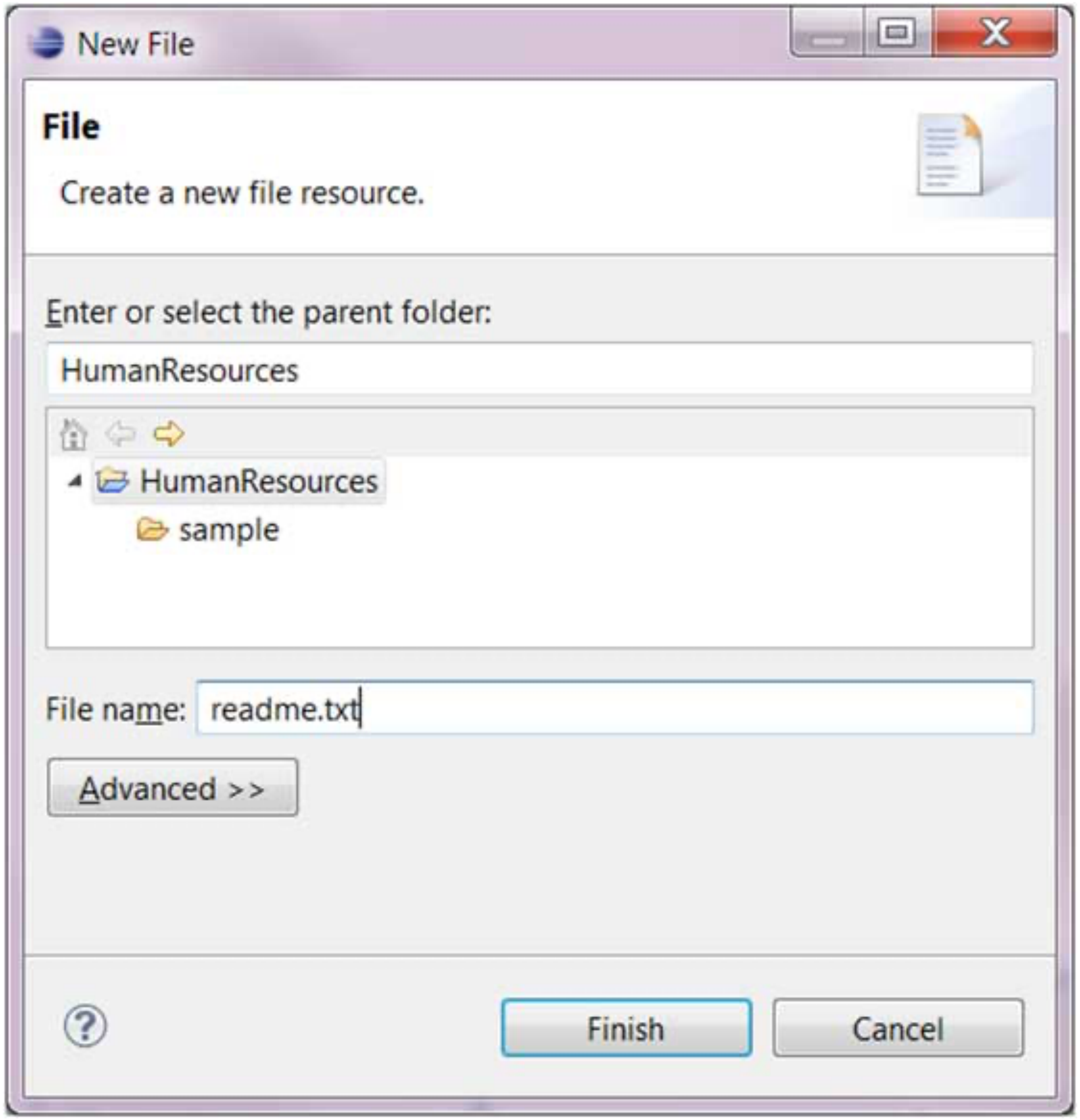
Shortcut	Description
✓ Annotation	Create an annotation type
✓ Class	Create a Java class
✓ Enum	Create an enum type
✓ Interface	Create a Java interface
✓ Java Project	Create a Java project
☐ Java Project from Existing Ant...	Create a Java project from an existing Ant
✓ Java Working Set	Creates a Java working set
✓ Package	Create a Java package
✓ Source Folder	Create a Java source folder

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.

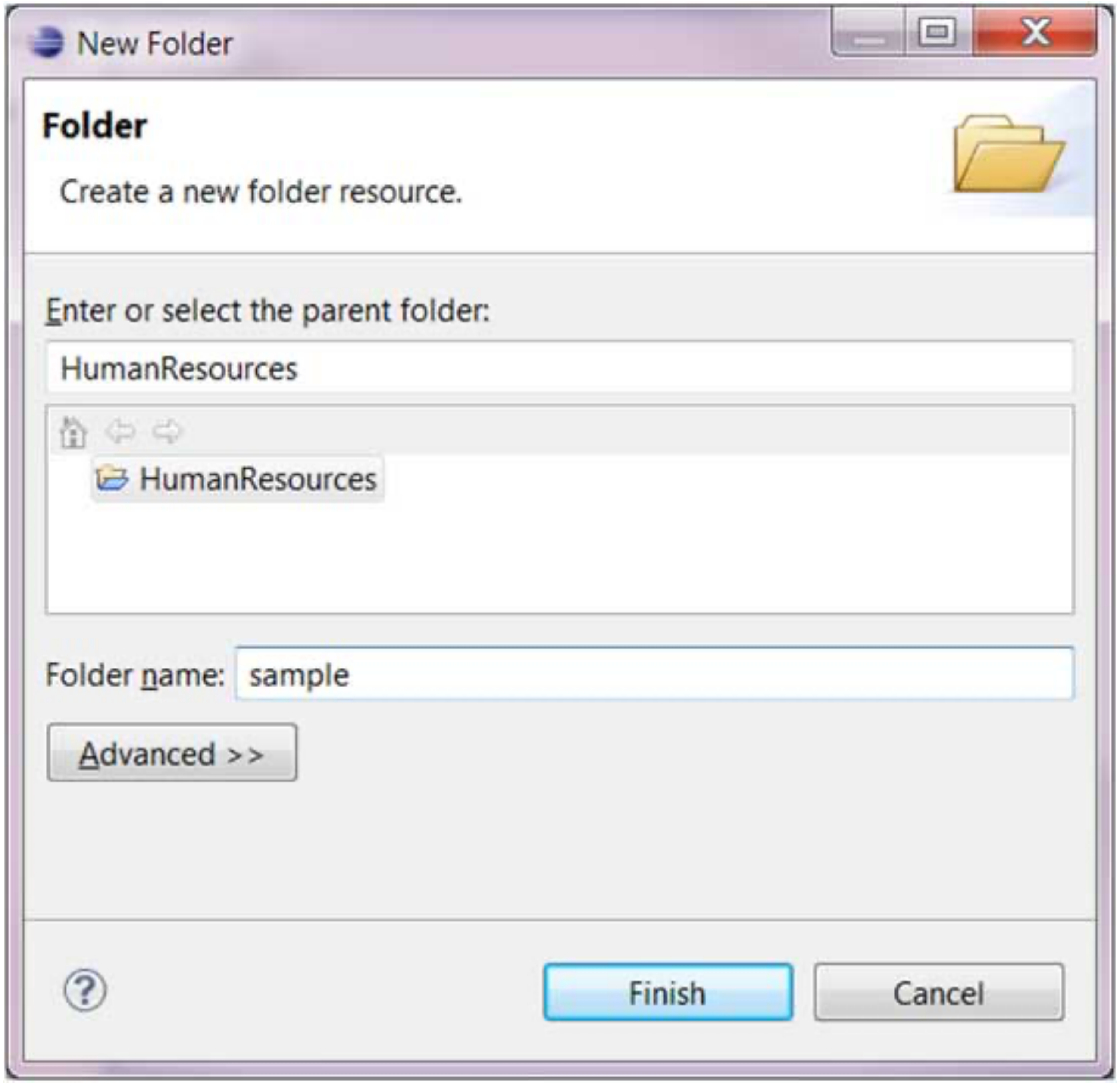


Enter or select the parent folder: HumanResources

HumanResources
sample

File name: readme.txt

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


New Folder

Folder

Create a new folder resource.

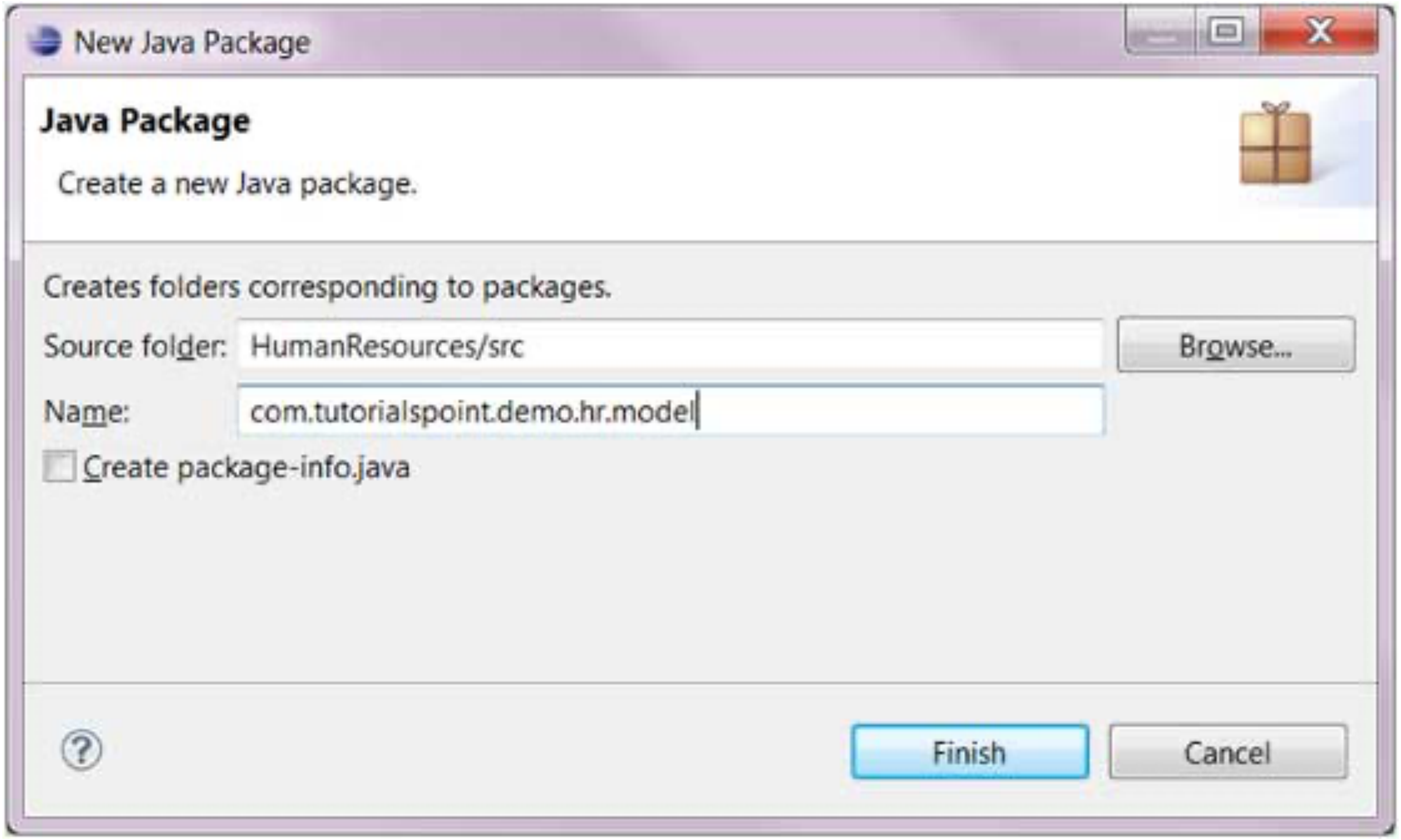
Enter or select the parent folder: HumanResources

HumanResources

Folder name: sample

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 Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

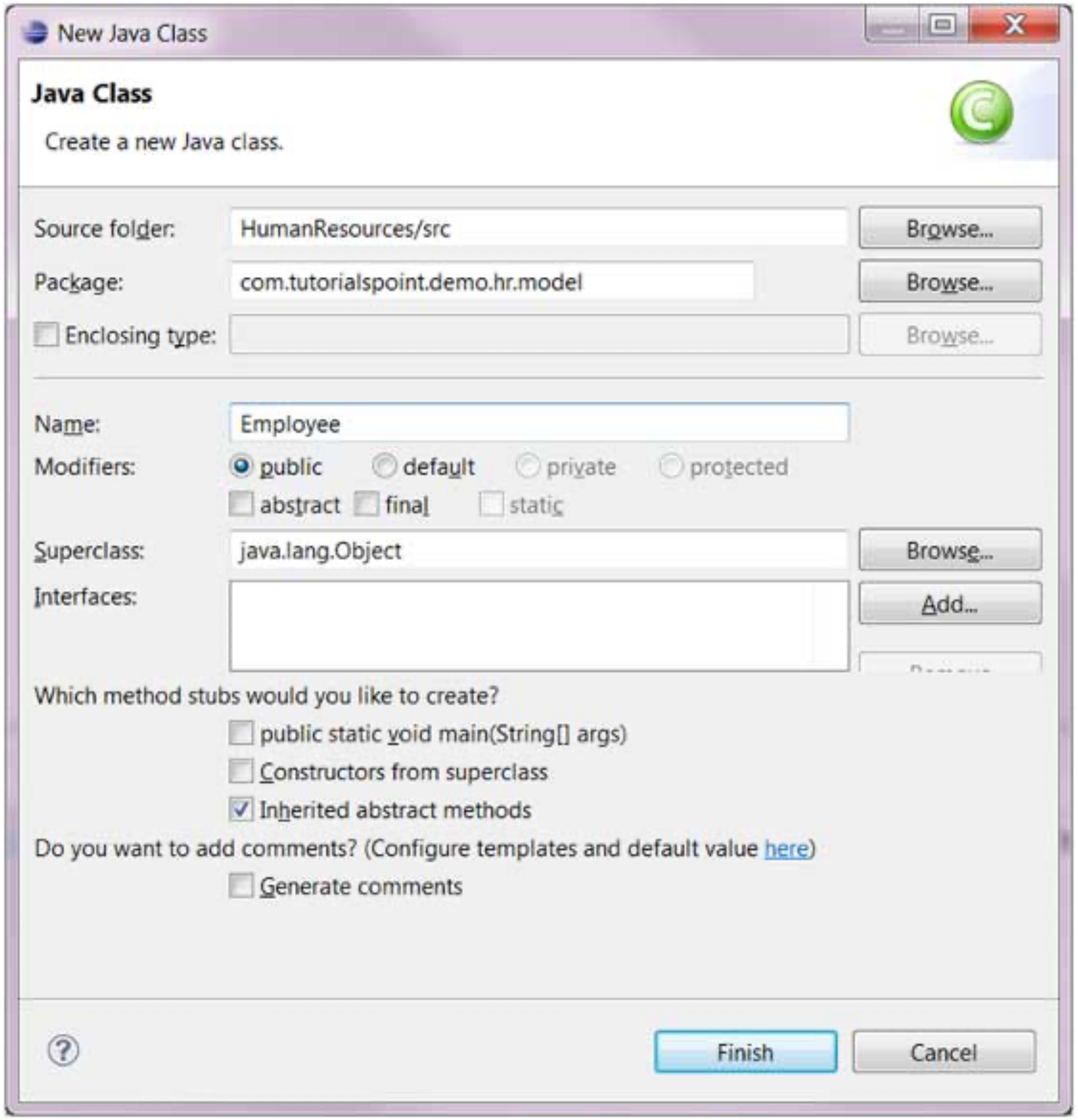
Source folder: HumanResources/src

Name: com.tutorialspoint.demo.hr.model

Create package-info.java

New Java Class Wizard

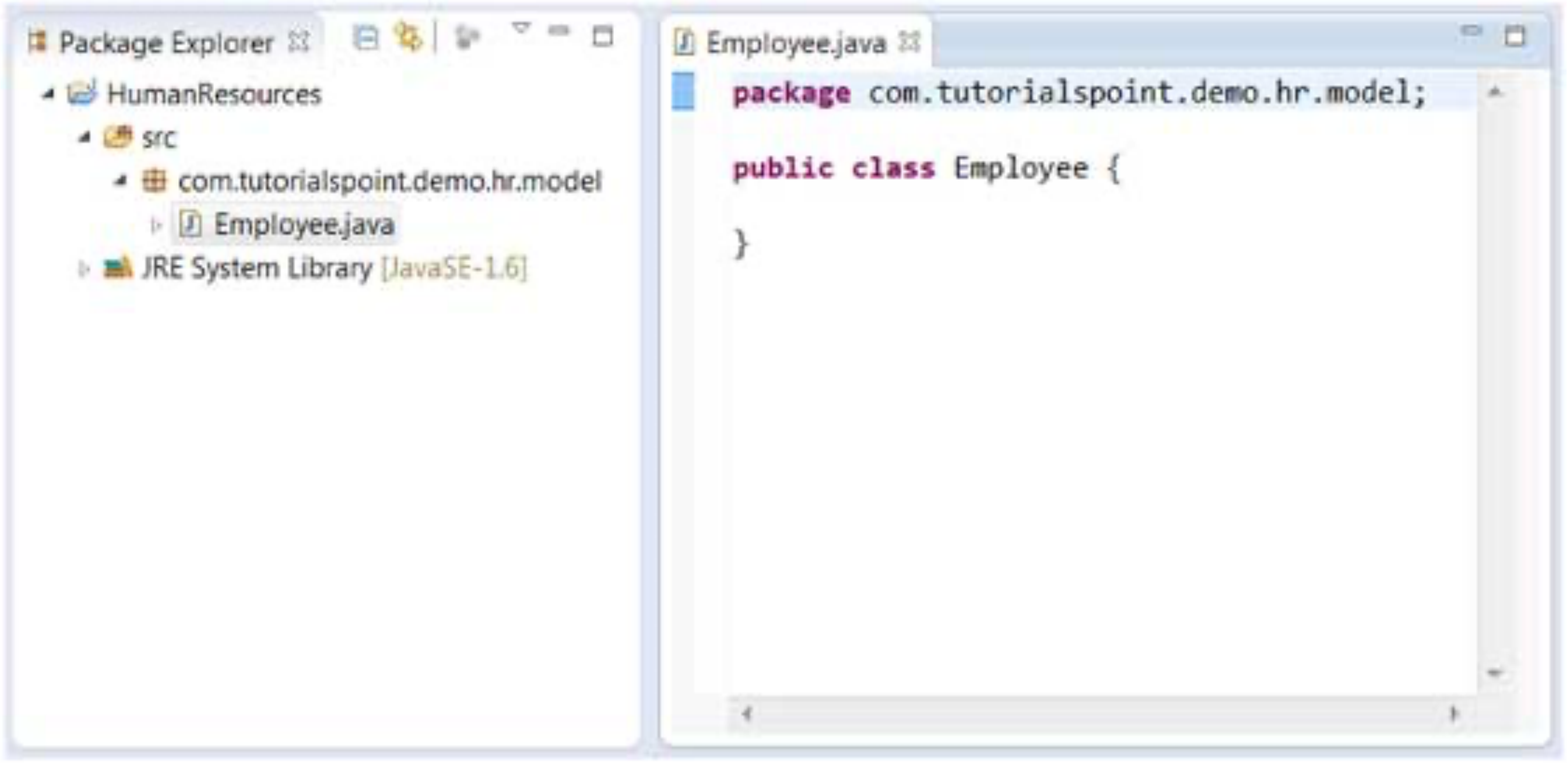
- By clicking on the File menu and selecting New → Class.
- By right clicking in the package explorer and selecting New → Class.
- By clicking on the class drop down button ( ) and selecting class ( ).



Source folder: HumanResources/src
Package: com.tutorialspoint.demo.hr.model
☐ Enclosing type:
Name: Employee
Modifiers: ● public ○ default ○ private ○ protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
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.


Package Explorer
HumanResources
src
com.tutorialspoint.demo.hr.model
Employee.java
JRE System Library [JavaSE-1.6]
Employee.java

```
package com.tutorialspoint.demo.hr.model;

public class Employee {

}
```

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/rlpfeeback**
- **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);
 }
}
```