Java Fundamentals

Training Module

Table of Contents

Chapter 1: Object Oriented Concepts

Introduction to OOP Concepts

Objects

Objects are key to understanding *object-oriented* technology. Look around right now and you'll find many examples of real-world objects: your dog, your desk, your television set, your bicycle.

Real-world objects share two characteristics: They all have

Object	State	Behavior
Dogs	name, color, breed, hungry	barking, fetching, wagging tail
Bicycles	current gear, current pedal cadence, current speed	changing gear, changing pedal cadence, applying brakes
Lamp	*on* or *off*	turn on and turn off lamp

Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

Take a minute right now to observe the real-world objects that are in your immediate area. For each object that you see, ask yourself two questions:

1. "What possible states can this object be in?"
2. "What possible behavior can this object perform?"

These real-world observations all translate into the world of object-oriented programming.

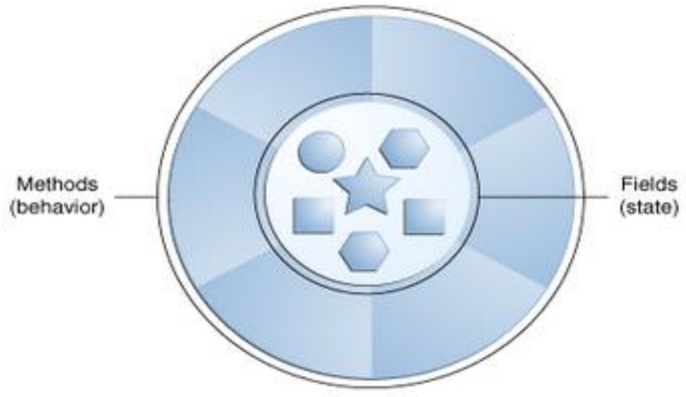
Java Training Module

Software objects are conceptually like real-world objects: they too consist of state and related behavior. An object stores its state in ***fields*** (variables in some programming languages) and exposes its behavior through ***methods*** (functions in some programming languages).

Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication.

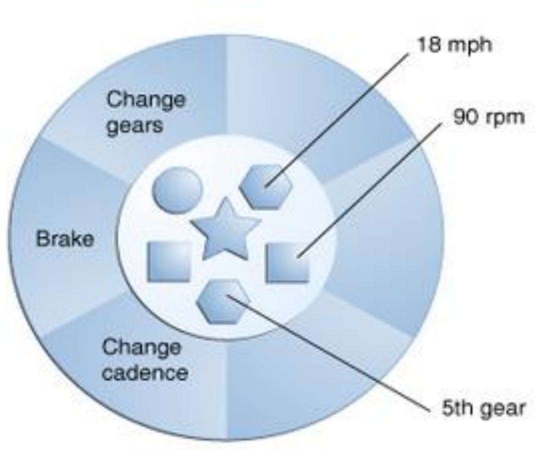
Hiding internal state and requiring all interaction to be performed through an object's methods is known as *data encapsulation* — a fundamental principle of object-oriented programming.

Consider a bicycle, for example:



A software object.

Java Training Module



A bicycle modeled as a software object.

By attributing state

- current speed
- current pedal cadence
- current gear

Providing methods for

- changing that state,
- the object remains in control of how the outside world can use it.

For example, if the bicycle only has 6 gears, a method to change gears could reject any value that is less than 1 or greater than 6.

Class

A class is a user defined blueprint from which individual objects are created.

A class can contain any of the following variable types.

- **Local variables** – Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables** – Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that class.
- **Class variables** – Class variables are variables declared within a class, outside any method, with the static keyword.

In the real world, you'll often find many individual objects all the same kind. There may be thousands of other bicycles in existence, all the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles.

A *class* is the blueprint from which individual objects are created.

Example of Object and class:

The following Bicycle class is one possible implementation of a bicycle:

```
class Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence(int newValue) {
         cadence = newValue;
    }

    void changeGear(int newValue) {
         gear = newValue;
    }

    void speedUp(int increment) {
         speed = speed + increment;
    }

    void applyBrakes(int decrement) {
         speed = speed - decrement;
    }

    void printStates() {
         System.out.println("cadence:" +
             cadence + " speed:" +
             speed + " gear:" + gear);
    }
}
```

The fields `cadence`, `speed`, and `gear` represent the object's state, and the methods (`changeCadence`, `changeGear`, `speedUp` etc.) define its interaction with the outside world.

You may have noticed that the `Bicycle` class does not contain a `main` method. That's because it's not a complete application; it's just the blueprint for bicycles that might be *used* in an application. The responsibility of creating and using new `Bicycle` objects belongs to some other class in your application.

Here's BicycleDemo class that creates two separate Bicycle objects and invokes their methods:

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

        // Create two different
        // Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on
        // those objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike1.printStates();

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
        bike2.printStates();
    }
}
```

The output of this test prints the ending pedal cadence, speed, and gear for the two bicycles:

```
cadence:50 speed:10 gear:2
cadence:40 speed:20 gear:3
```

Elements of OOP

The element of OOP is

- Inheritance
- Encapsulation
- Association
- Abstraction
- Polymorphism

Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance, the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

Different kinds of objects often have a certain amount in common with each other. Mountain bikes, road bikes, and tandem bikes, for example, all share the characteristics of bicycles (current speed, current pedal cadence, current gear). Yet each also defines additional features that make them different: tandem bicycles have two seats and two sets of handlebars; road bikes have drop handlebars; some mountain bikes have an additional chain ring, giving them a lower gear ratio.

Object-oriented programming allows classes to *inherit* commonly used state and behavior from other classes. In this example, Bicycle now becomes the *superclass* of MountainBike, RoadBike, and TandemBike. In the Java programming language, each class can have one direct superclass, and each superclass has the potential for an unlimited number of *subclasses*

Encapsulation

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java.

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

Association

Association establishes relationship between two separate **classes** through their **objects**. The relationship can be one to one, One to many, many to one and many to many. It refers to how objects are related to each other and how they are using each other's functionality. Composition and aggregation are two types of association.

Composition

The composition is the strong type of association. An association is said to composition if an Object owns another object and another object cannot exist without the owner object. Consider the case of Human having a heart. Here Human object contains the heart and heart cannot exist without Human.

Aggregation

Aggregation is a weak association. An association is said to be aggregation if both Objects can exist independently. For example, a Team object and a Player object. The team contains multiple players, but a player can exist without a team.

Abstraction

Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

Test Your Knowledge

1. Which among the following is false?

A. Object must be created before using members of a class
B. Memory for an object is allocated only after its constructor is called
C. Objects can’t be passed by reference
D. Objects size depends on its class data members

2. The object can’t be:

A. Passed by reference
B. Passed by value
C. Passed by copy

D. Passed as function

3. Which was the first purely object-oriented programming language developed?

A. Java
B. C++
C. SmallTalk
D. Kotlin

4. Which of the following best defines a class?

A. Parent of an object
B. Instance of an object
C. Blueprint of an object
D. Scope of an object

5. What do you call the languages that support classes but not polymorphism?

A. Class based language
B. Procedure Oriented language
C. Object-based language
D. If classes are supported, polymorphism will always be supported

Chapter 2:Overview of JAVA Platform

Introduction to Java Programming Language

Brief History of JAVA

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of java starts with Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were

- Simple, Robust
- Portable
- Platform-independent
- Secured
- High Performance
- Multithreaded
- Architecture Neutral
- Object-Oriented
- Interpreted and Dynamic

James Gosling is the founder of java

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given the significant points that describe the history of Java.

Java History from Oak to Java

James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team. Originally designed for small, embedded systems in electronic appliances like set-top boxes. Firstly, it was called "Greentalk" by James Gosling, and file extension was .gt. After that, it was called Oak and was developed as a part of the Green project.

Why Java named "Oak"? Why Oak?

Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania, etc. In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.

Why Java Programming named "Java"? Why had they chosen java name for java language?

The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling, "Java was one of the top choices along with Silk". Since Java was so unique, most of the team members preferred Java than other names.

Java is an island of Indonesia where first coffee was produced (called java coffee). Notice that Java is just a name, not an acronym. Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995. In 1995, Time magazine called Java one of the Ten Best Products of 1995. JDK 1.0 released in (January 23, 1996).

Eclipse

Introduction to Eclipse IDE

Eclipse is now controlled by an independent nonprofit organization called the Eclipse Foundation. Since 2001, it has been downloaded over 50 million times.

Eclipse is an Integrated Development Environment for

- “anything, and nothing at all,” meaning that it can be used to develop software in any language, not just Java.
- a proprietary replacement for Visual Age for Java from IBM but was open sourced in November 2001.
- being used by thousands of developers worldwide.
- classes on programming and object-oriented design.

System Requirements

Eclipse runs on today’s most popular operating systems, including Windows XP, Linux, and Mac OS X. It requires Java to run, so if you don’t already have Java installed on your machine, you must first install a compatible java version for Eclipse.

Exercise 1: Setup Java

To implement Java application, you can install Java in windows or Linux operating system. Also, you can use Eclipse IDE to handle all levels of Java projects development.

Following are steps to download eclipse IDE or Java on windows or Linux operating system.

To install java in windows system, download the Java v8.

Setting environment variable

Procedure

1. Download or save the appropriate JDK version for Windows. Assuming you have installed Java in c:\Program Files\java\jdk directory
2. Right-click the Computer icon on your desktop and select **Properties**.
3. Click **Advanced system settings**.
4. Click **Environment variables**.
5. Under User variables, click **New**.
6. Enter **JAVA_HOME** as the variable name.
7. Edit the path to the JDK as the variable value. For Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then edit your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'

Note: do not erase or delete the existing path, only add the new <installed java> path.

To install java in Unix/Linux system, *untar the .tar.gz* file.

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation, if you have trouble doing this.

Example, if you use *bash* as your shell, then you would add the following line to the end of your '.bashrc: export PATH = /path/to/java:$PATH'

To setup eclipse IDE

1. Use https://www.eclipse.org/downloads/ to download eclipse IDE.

The capabilities of each packaging of eclipse are different. Java developers typically use Eclipse Classic or Eclipse IDE for developing Java application.

Exercise 2: Implement first Java application using command prompt

In this exercise, you will develop a simple java application, which will print Welcome to Java Application.

Instructions

1. Open any editor tool such as Notepad, Notepad++.
2. Enter following code.

```
public class Welcome{

   /* This is my first java program.
    * This will print 'Welcome to Java Application' as the output
    */

   public static void main(String []args) {
      System.out.println("Welcome to Java Application");
   }
}
```

3. Save file as “Welcome.java” in your directory “C:\StudentFolder”
4. Open a command prompt window and go to the directory where you saved the class. Assume it's “C:\StudentFolder”.
5. Enter **javac Welcome.java** and press enter to compile your code.

If there are no errors in your code, the command prompt will take you to the next line (Assumption: The path variable is set).

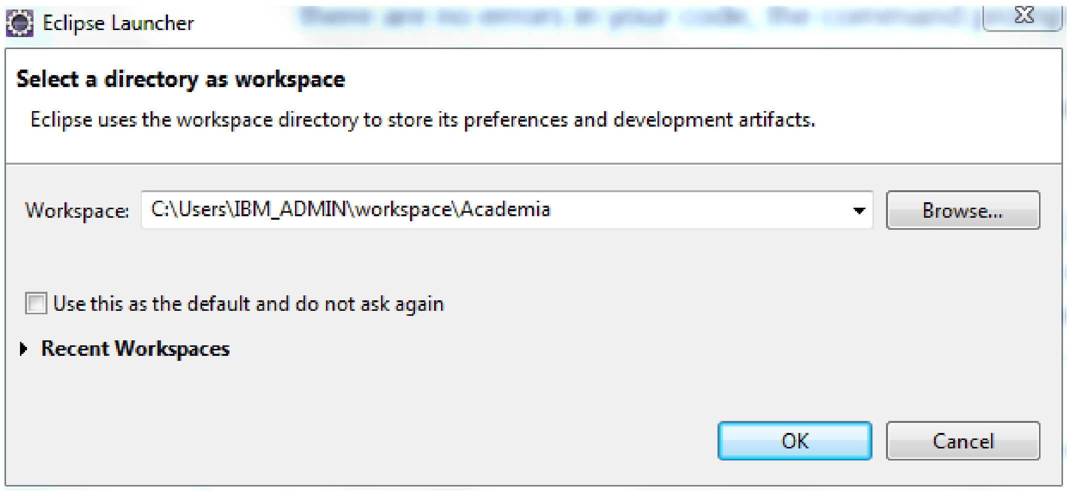
6. Enter **java Welcome** to run your program. You will be able to see '**Welcome to Java Application'** printed on the window.

Exercise 3: Implement Java application using eclipse IDE

In this exercise, you will develop a simple java application, which will print test "Welcome to Eclipse Integrated Development Environment".

Instructions

1. Double the eclipse IDE tool to view the screen as shown in the screen capture.



Select a directory as workspace

Eclipse uses the workspace directory to store its preferences and development artifacts.

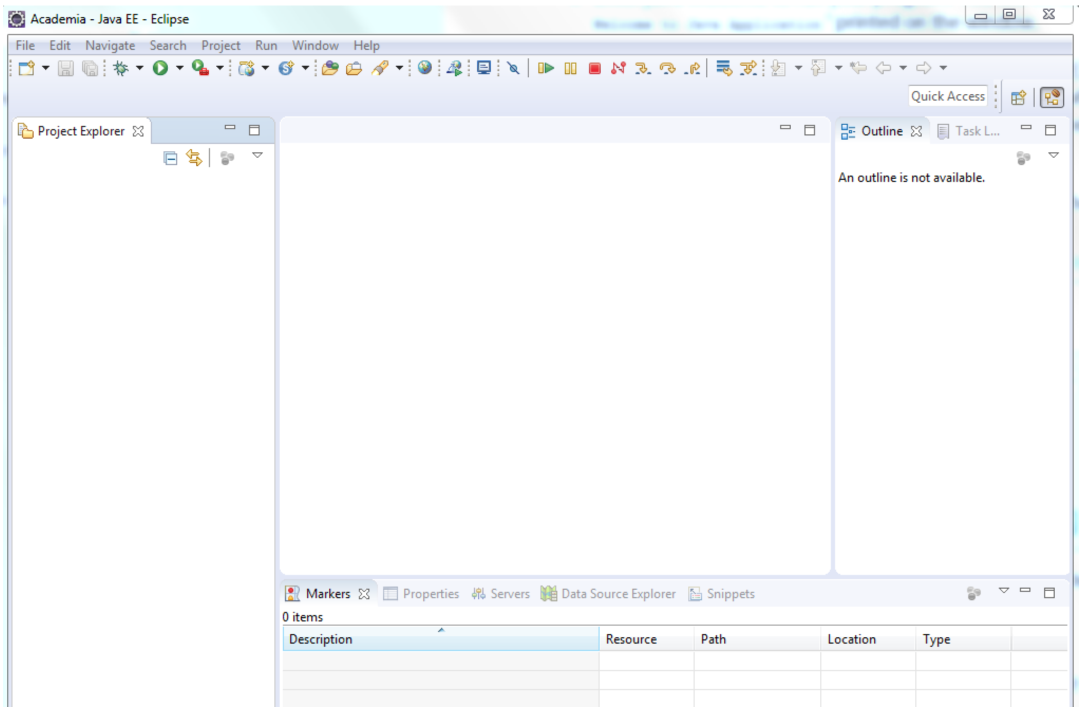
Workspace: C:\Users\IBM_ADMIN\workspace\Academia

Use this as the default and do not ask again

Recent Workspaces

2. Create your own workspace, for Example, **Academia** is created for the demo purpose in this exercise.
3. Click **OK**.
4. Click **close icon** of welcome button to view the complete view of the eclipse IDE.

Java Training Module

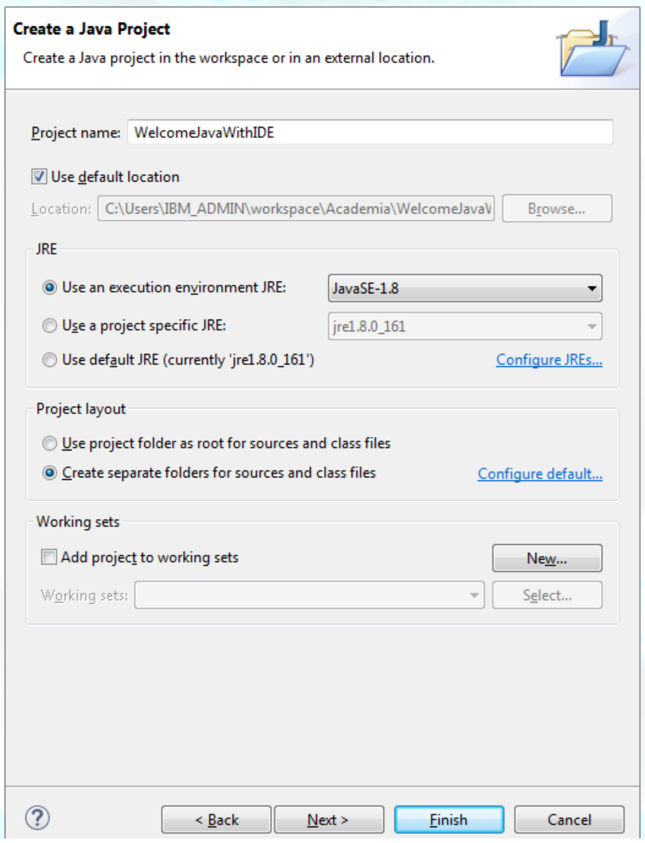


An outline is not available.

0 items

Description	Resource	Path	Location	Type

5. Click **File** > **Project** > **Java** > **Java Project**
6. Click **Next**.
7. Enter **Project name** as **WelcomeJavaWithIDE** as shown in the screen capture.



Create a Java Project

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

Project name: WelcomeJavaWithIDE

☑ Use default location

Location: C:\Users\IBM_ADMIN\workspace\Academia\WelcomeJava\

JRE

● Use an execution environment JRE: JavaSE-1.8

○ Use a project specific JRE: jre1.8.0_161

○ Use default JRE (currently 'jre1.8.0_161')

Project layout

○ Use project folder as root for sources and class files

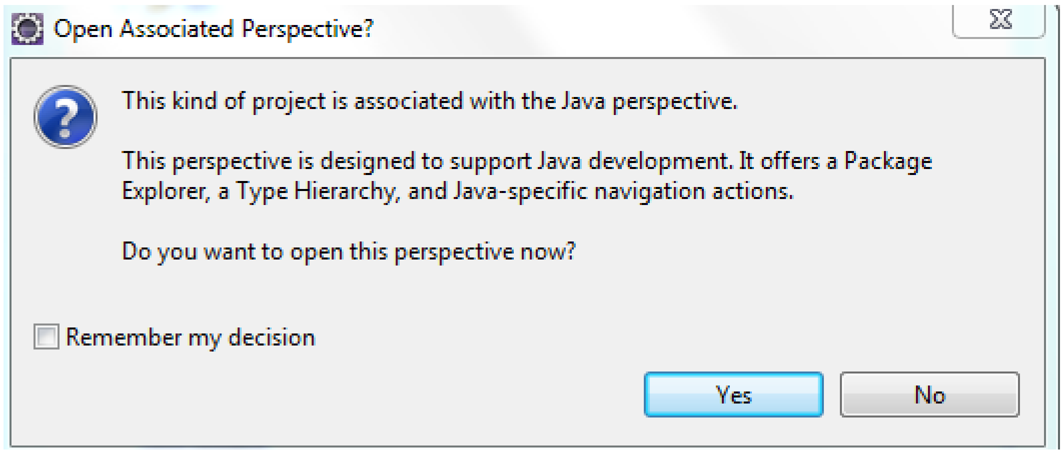
● Create separate folders for sources and class files

Working sets

☐ Add project to working sets

Working sets:

8. Click **Next**.
9. Click **Finish**, you will see Open Associated Perspective dialog box.



Open Associated Perspective?

This kind of project is associated with the Java perspective.

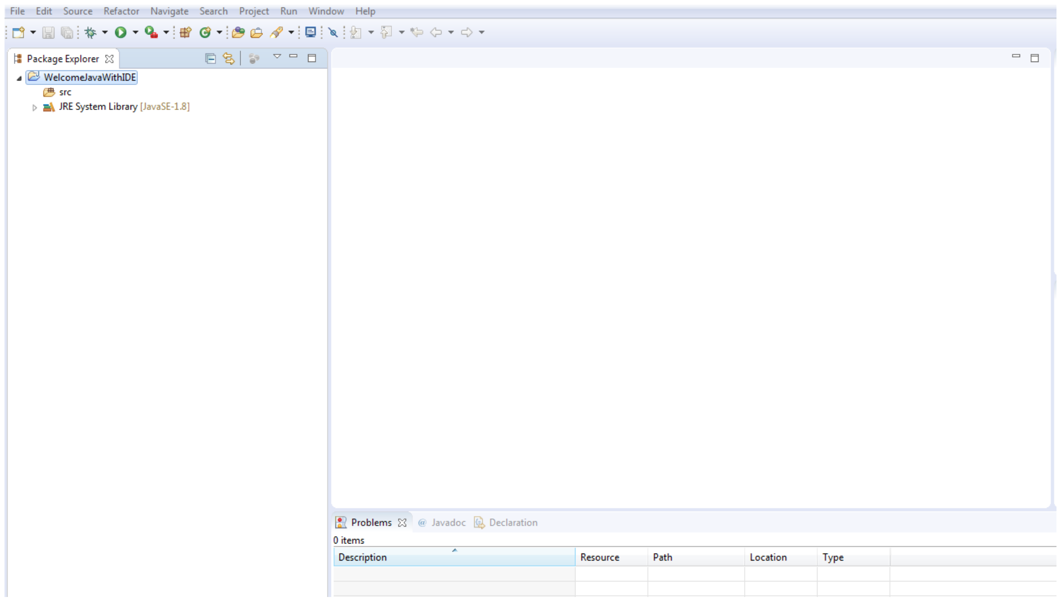
This perspective is designed to support Java development. It offers a Package Explorer, a Type Hierarchy, and Java-specific navigation actions.

Do you want to open this perspective now?

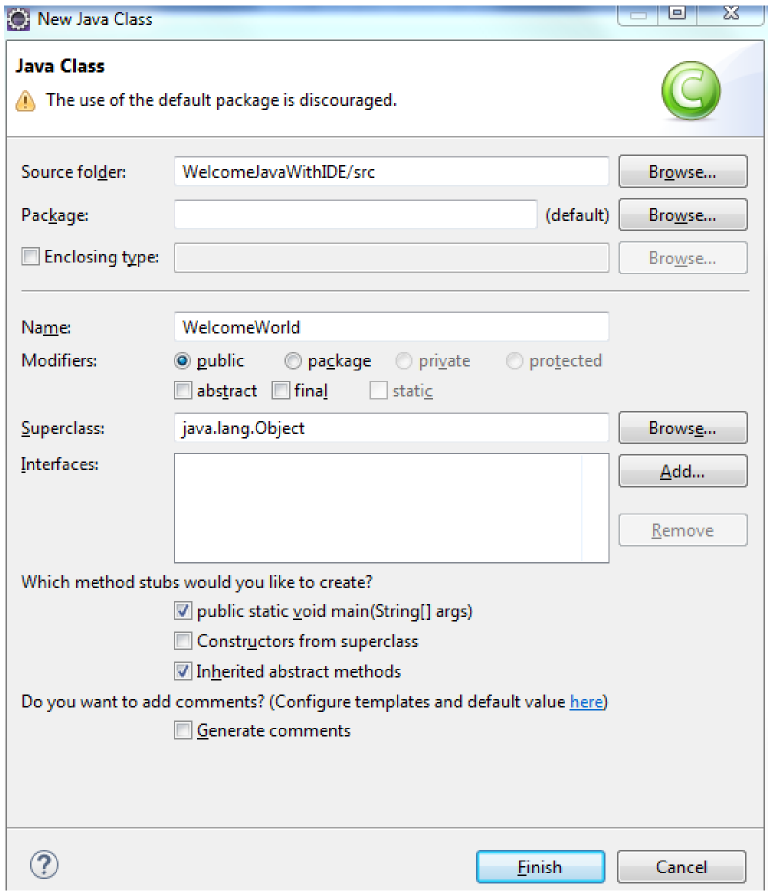
☐ Remember my decision

10. Click **Yes** to view the user interface as shown in the screen capture.

Java Training Module



11. **Right Click** WelcomeJavaWithIDE.
12. Click **New** > **Class**
13. In Java Class page, enter
    a. **Name** as **WelcomeWorld**
    b. Check **public static void main(String []args)** option as shown in the screen capture.


New Java Class

Java Class

The use of the default package is discouraged.

Source folder: WelcomeJavaWithIDE/src

Package: (default)

Enclosing type: [ ]

Name: WelcomeWorld

Modifiers: [✓ public] package private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

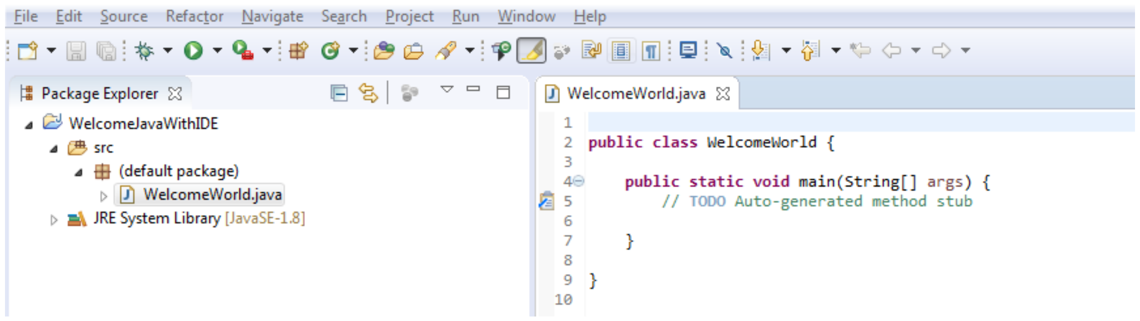
[ ] Constructors from superclass

[✓] Inherited abstract methods

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

[ ] Generate comments

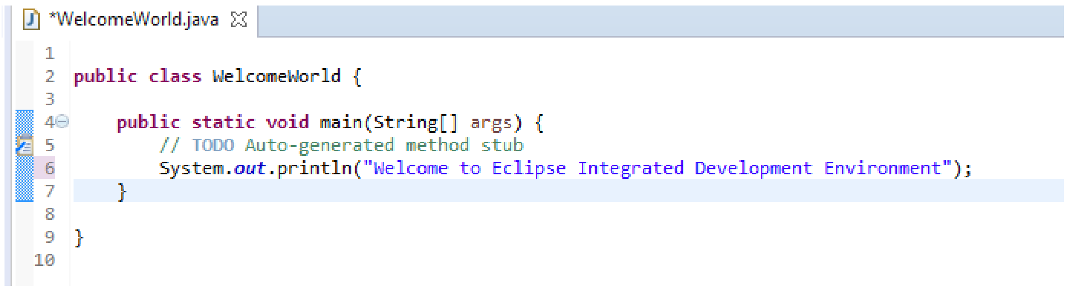
14. Click **Finish** to view the code as shown in the screen capture.



File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
WelcomeJavaWithIDE
src
(default package)
WelcomeWorld.java
JRE System Library [JavaSE-1.8]
WelcomeWorld.java

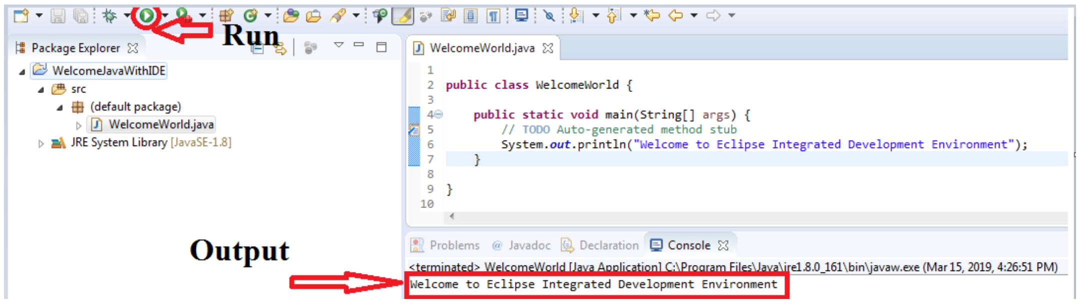
```

public class WelcomeWorld {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}

```

15. Enter the code with the print statement as shown in the screen capture.



```java

public class WelcomeWorld {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Welcome to Eclipse Integrated Development Environment");
    }

}
```

16. Click **Run** icon from the tool bar and select WelcomeWorld.java in Save and Launch page, to view the result of the application as shown in the screen capture.



```java

public class WelcomeWorld {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Welcome to Eclipse Integrated Development Environment");
    }

}
```

```
<terminated> WelcomeWorld [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 15, 2019, 4:26:51 PM)
Welcome to Eclipse Integrated Development Environment
```

Command line arguments

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

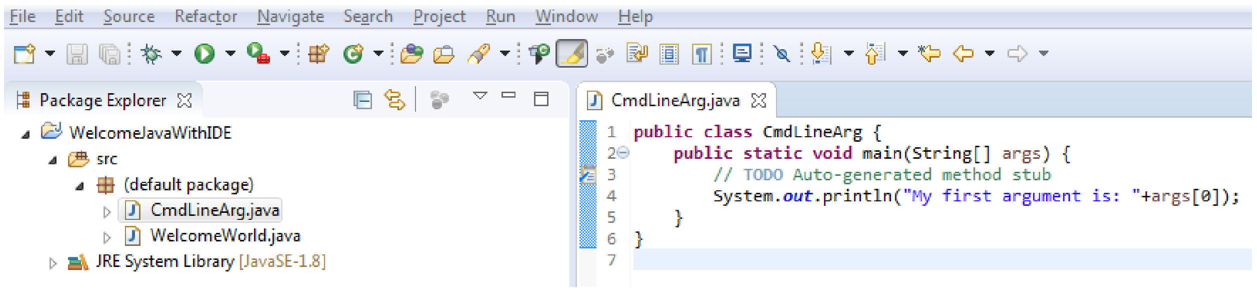
So, it provides a convenient way to check the behavior of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.

Exercise 4: Implement command line argument using eclipse IDE

In this exercise you will learn how to develop a command line argument Java program.

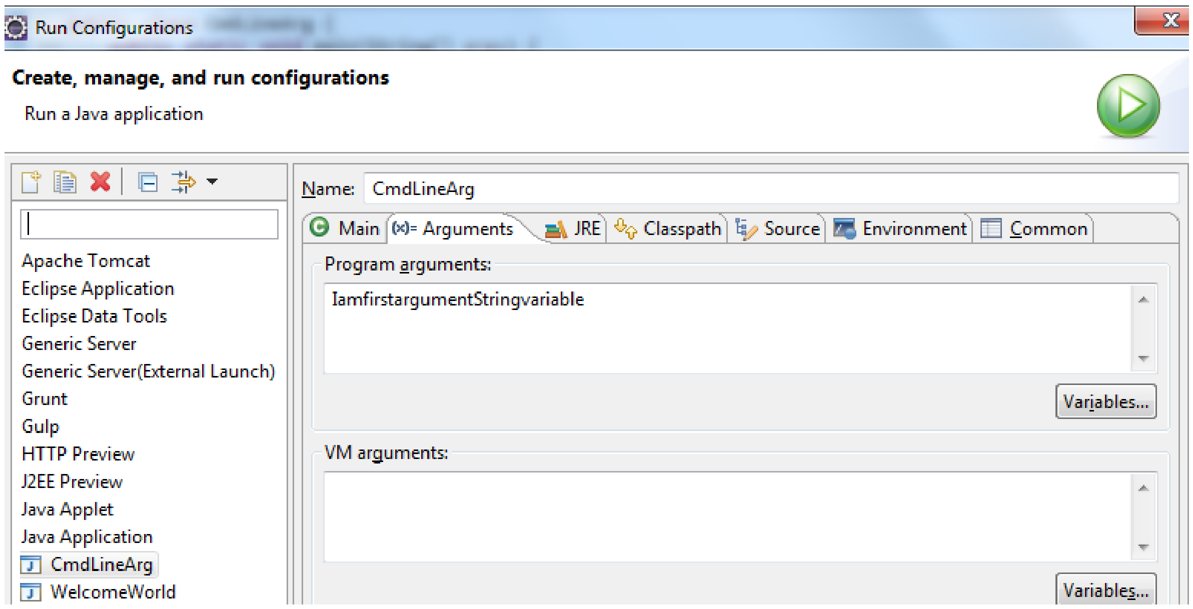
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **CmdLineArg** and enter the code as shown in the screen capture.



```java
public class CmdLineArg {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("My first argument is: "+args[0]);
    }
}
```

5. Select **CmdLineArg.java** from Package Explorer.
6. Click **Run** > **Run Configurations**.
7. Select **Arguments** tab.
8. In Program arguments, enter **IamfirstargumentStringvariable** without space, as shown in the screen capture.



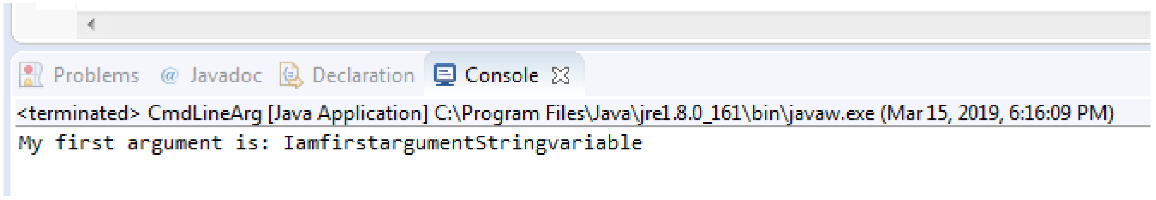
Name: CmdLineArg

Program arguments:

IamfirstargumentStringvariable

VM arguments:

9. Click **Run** to view the output as shown in the screen capture.



```
<terminated> CmdLineArg [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 15, 2019, 6:16:09 PM)
My first argument is: IamfirstargumentStringvariable
```

Similarly, you can use String []args, command line variable to define multiple arguments. Multiple arguments can be specified as [5]args, each argument is separated by a space.

Test Your Knowledge

1. What are activities you can do in Eclipse?

   A. Create generic projects
   B. Edit files in a generic text editor
   C. Share files and project in a CVS (Concurrent Version System) server
   D. All the above

2. What are the Basic things an IDE includes?

   A. A source code editor
   B. A compiler and/or an interpreter
   C. Build automation tools
   D. All of these

3. What are the steps to change the JDK compliance level?

   A. Go to Windows
   B. Select Preferences
   C. Select Java – Compiler
   D. All of these

4. Who is founder of JAVA

   A. James Gosling
   B. Mike Sheridan
   C. Patrick Naughton
   D. All the above

Java Training Module

5. What is command to check JAVA version

   A. Java -version
   B. Java –version
   C. Java cp version
   D. All the above

Chapter 3 -Java Language Fundamentals

Datatypes, Variables and Operators

Datatypes

Values that variable can hold is called datatypes. There are 2 types of datatypes

1. Primitive Datatypes
   There are 8 types of primitive datatypes.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,808
float	4 bytes	Sufficient for storing 6-7 decimal digits
double	8 bytes	Sufficient for storing 15 decimal digits
boolean	1 byte	Sufficient for storing 15 decimal digits
char	2 bytes	Stores a single character/letter

2. Non-Primitive Datatypes
Non-primitive datatypes are those which uses primitive datatype as base like array, enum, class etc.

Type Casting

Assigning a value of one type of variable to another type is called Type Casting.

Java Training Module

Syntax

dataType variableName = (dataType) variableToConvert;

There are two types of type casting

1. Widening casting (implicit) : - Automatic type casting take place when target type is larger than source type

   byte→ short → int → long→ float→double

2. Narrowing casting(explicit) :- When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.
   Double→ float→long→int→short→ byte

Variables

Variable are containers that stores the value while the java program is executed. There are 3 types of variables.

1. Local variable – Variable declare within method is called local variable.
2. Instance variable – variable declares within class but outside method is called instance variable
3. Static variable – Static variable is a class level variable. A single copy of static variable is created and shared among all the instances of the class. Memory allocation happens once when class loads in memory.

Operators

Operators are used to perform operations on variables and values.

The types of operators are:

1. Arithmetic operator
2. Shift operator
3. Relational operator
4. Bitwise operator
5. Logical operator
6. Ternary operator
7. Assignment operator
8. Unary operator

1. Unary Operators

   Unary operators need only one operand. They are used to increment, decrement or negate a value.

Operator	Description
+	Unary plus
-	Unary Minus
++	Increment operator
--	Decrement operator
!	Logical not operator

2. Arithmetic Operators
   Arithmetic operators are used to perform common mathematical operations.

Operator	Description
*	Multiplication
+	Addition
-	Subtraction
/	Division
%	Modulo

3. Shift operator

The Java left shift operator << is used to shift all the bits in a value to the left side of a specified number of times.

Operator	Description
<<	Left shift operator
>>	Right shift operator
>>>	Unsigned right shift operator

4. Relational operator

Relational operators are used to compare two values.

Operator	Description
==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

5. Bitwise Operator

Bitwise and bit shift operators are used on integral types (byte, short, int and long) to perform bit-level operations.

Operator	Description
&	Bitwise AND operator
|	Bitwise OR operator
~	Bitwise complement operator
^	Bitwise XOR Operator

6. Logical Operators

These operators are used to perform "logical AND" and "logical OR" operation.

Operator	Description
&&	Logical AND
||	Logical OR

7. Ternary Operators

Ternary operator is a shorthand version of if-else statement.

```
Syntax condition ? if true : if false
```

The statement means that if the condition evaluates to true, then execute the statements after the '?' else execute the statements after the ':'.

8. Assignment Operators

'=' Assignment operator is used to assign a value to any variable. It has a right to left associativity,

In many cases assignment operator can be combined with other operators to build a shorter version of statement called Compound Statement.

Operator	Description
+=	adding left operand with right operand and then assigning it to variable on the left
-=	subtracting left operand with right operand and then assigning it to variable on the left.
*=	multiplying left operand with right operand and then assigning it to variable on the left.
/=	dividing left operand with right operand and then assigning it to variable on the left.
%=	assigning modulo of left operand with right operand and then assigning it to variable on the left.

Scanner Class

Scanner class is used to get user input. It is found in java.util package.

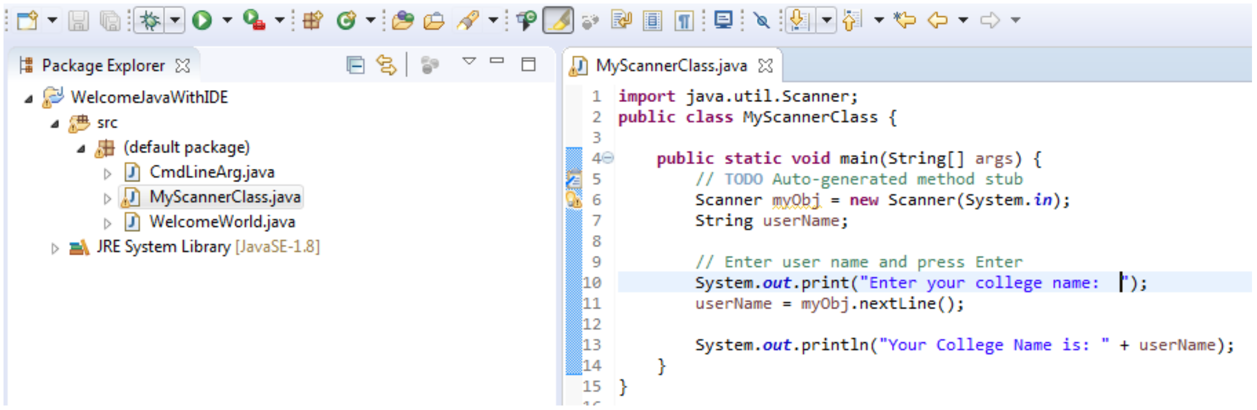
Method	Description
nextLine()	Reads String value from user.
nextFloat()	Reads value from user.
nextBoolean()	Reads boolean value from user.
nextByte()	Reads byte value from user.
nextDouble()	Reads double value from user.
nextInt()	Reads integer value from user.
nextLong()	Reads long value from user.
nextShort()	Reads short value from user.

Exercise 1: Implement scanner class

In this exercise you will learn how to get user input through Scanner class.

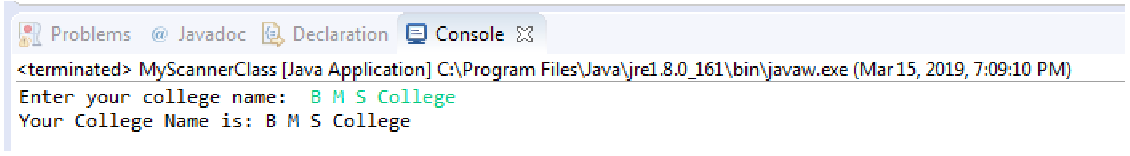
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyScannerClass** and enter the code as shown in the screen capture.



```
import java.util.Scanner;
public class MyScannerClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner myObj = new Scanner(System.in);
        String userName;

        // Enter user name and press Enter
        System.out.print("Enter your college name:  ");
        userName = myObj.nextLine();

        System.out.println("Your College Name is: " + userName);
    }
}
```

5. Select **MyScannerClass.java** from Package Explorer.
6. Click **Run**.
7. In the **Console** tab, Enter **B M S College** against Enter your college name: to view the output as shown in the screen capture.



```
<terminated> MyScannerClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 15, 2019, 7:09:10 PM)
Enter your college name:  B M S College
Your College Name is: B M S College
```

Programming Constructs

All programming language utilize program constructs. They are used to control the order (flow) in which statements are executed (or not executed).

A selection statement provides for selection between alternatives. We can identify three types of selection construct:

- If statements
- Case statements
- Looping

The details of these statements are covered in detail:

If statement

If statement is used to test the statement. It checks boolean condition: true or false.

Java has the following conditional statements:

Condition	Statement
If statement	to specify a block of code to be executed, if a specified condition is true.
If-else statement	to specify a block of code to be executed, if the same condition is false.
If-else-if statement	to specify a new condition to test, if the first condition is false.
Nested if statement	To place if statement inside another IF Statement.

Syntax

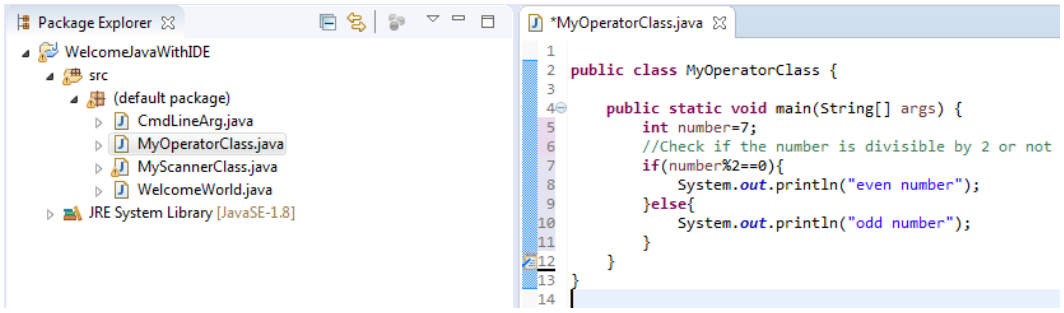
```
if (condition)
{
   // Executes this block if
   // condition is true
}
else
{
   // Executes this block if
   // condition is false
}
```

Exercise 2: Implement If statement

In this exercise you will learn how to check the even number using java operators to implement if statements in a class.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyOperatorClass** and enter the code as shown in the screen capture.



```

public class MyOperatorClass {

    public static void main(String[] args) {
        int number=7;
        //Check if the number is divisible by 2 or not
        if(number%2==0){
            System.out.println("even number");
        }else{
            System.out.println("odd number");
        }
    }
}

```

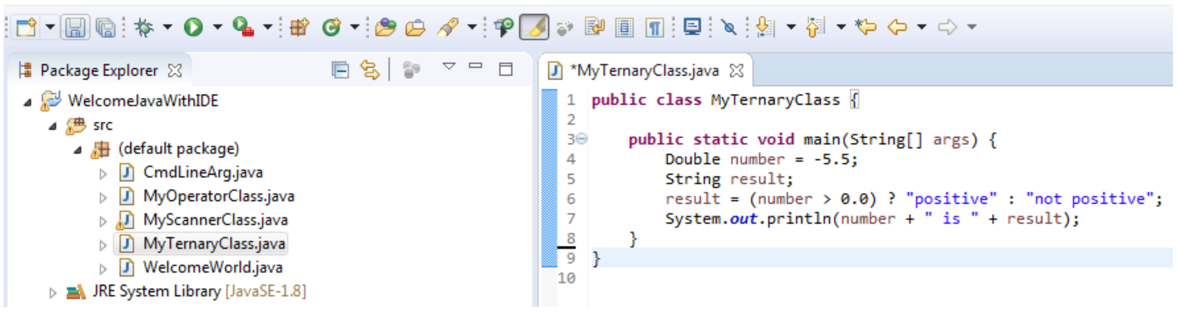
5. Select **MyOperatorClass.java** from Package Explorer.
6. Click **Run** to view the output **odd number** in the **console** tab.

Exercise 3: Implement ternary operator

In this exercise you will learn how to use ternary operator.

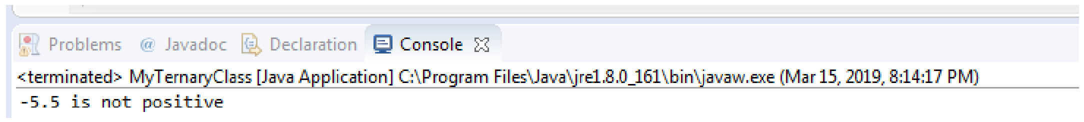
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next.**
4. Enter Project name as **MyTernaryClass** and enter the code as shown in the screen capture.



```
public class MyTernaryClass {

    public static void main(String[] args) {
        Double number = -5.5;
        String result;
        result = (number > 0.0) ? "positive" : "not positive";
        System.out.println(number + " is " + result);
    }
}
```

5. Select **MyTernaryClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.



```
<terminated> MyTernaryClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 15, 2019, 8:14:17 PM)
-5.5 is not positive
```

Case statements

Generally, "if ... then ... else ..." statement supports selection from only two alternatives. You can of course nest such statements, but it is usually more succinct to use a case statement. Case statements allow selection from many alternatives where each alternative is linked to a predicate.

Switch Operator

- Tests a single variable for several alternative values and executes the corresponding case.
- Any case without break falls through
- Next case will also be executed
- default clause handles values not explicitly handled by a case

Syntax

Syntax

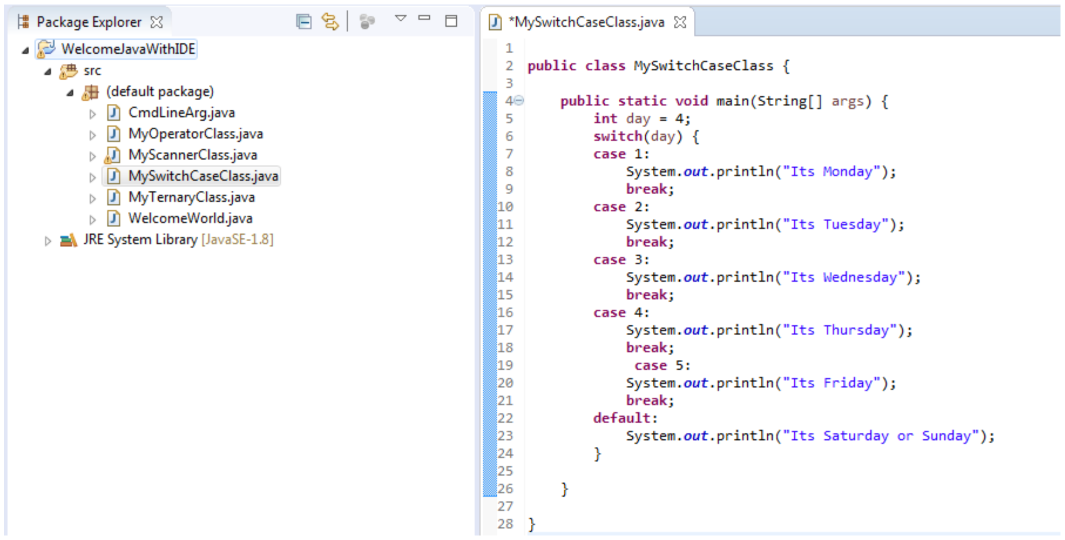
```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Exercise 4: Implement switch case

In this exercise you will learn how to implement switch case to identify the day of the week by an integer datatype.

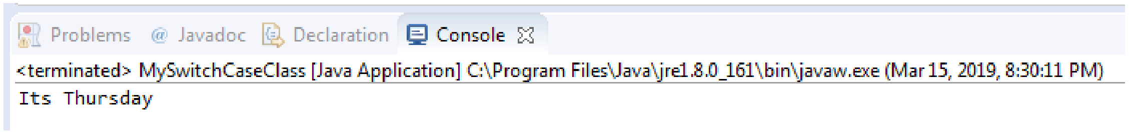
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MySwitchCaseClass** and enter the code as shown in the screen capture.



```

public class MySwitchCaseClass {

    public static void main(String[] args) {
        int day = 4;
        switch(day) {
        case 1:
            System.out.println("Its Monday");
            break;
        case 2:
            System.out.println("Its Tuesday");
            break;
        case 3:
            System.out.println("Its Wednesday");
            break;
        case 4:
            System.out.println("Its Thursday");
            break;
             case 5:
            System.out.println("Its Friday");
            break;
        default:
            System.out.println("Its Saturday or Sunday");
        }

    }

}
```

5. Select **MySwitchCaseClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.



```
<terminated> MySwitchCaseClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 15, 2019, 8:30:11 PM)
Its Thursday
```

Looping

A repetition construct causes a group of one or more program statements to be invoked repeatedly until some end condition is met.

Typically, such constructs are used to step through arrays or linked lists.

Loop	Statement	Syntax

While loop	Fixed count loops, that repeats a predefine number of times.	while (condition) {
		  // code block to be executed
		}

For loop	Variable count loops, that repeats an unspecified number of times.	for (statement 1; statement 2; statement 3) {
		  // code block to be executed
		}

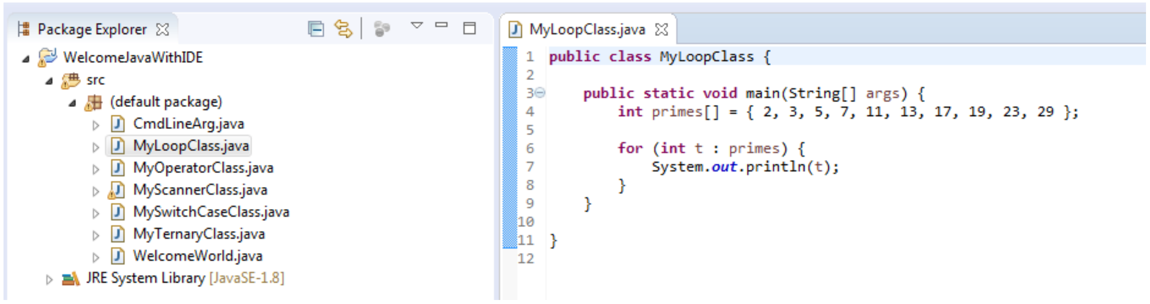
Do-while loop	Executes at least once and works as while loop.	do {
		  // code block to be executed
		}
		while (condition);

Exercise 5: Implement for loop

In this exercise you will learn how to implement loop to list the prime numbers.

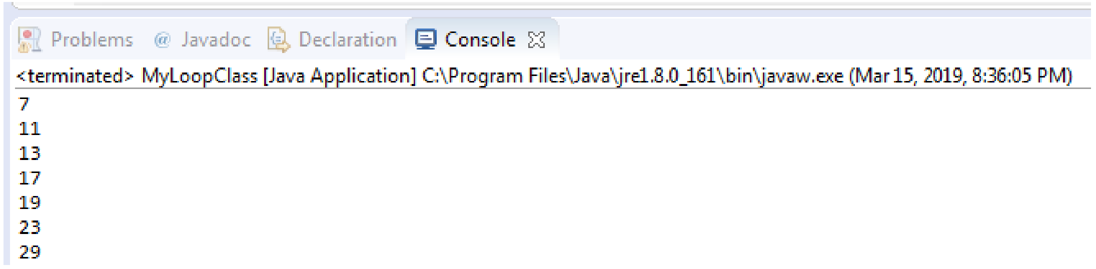
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyLoopClass** and enter the code as shown in the screen capture.



```
public class MyLoopClass {

    public static void main(String[] args) {
        int primes[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 };

        for (int t : primes) {
            System.out.println(t);
        }
    }

}
```

5. Select **MyLoopClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.

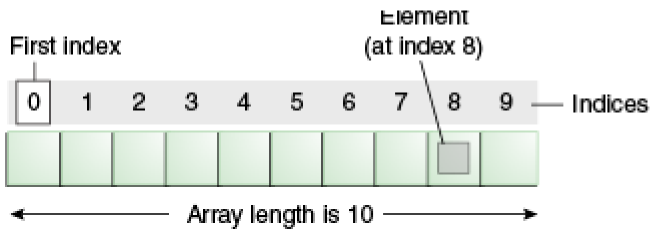


```
<terminated> MyLoopClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 15, 2019, 8:36:05 PM)
7
11
13
17
19
23
29
```

Arrays

Arrays in Java are homogeneous data structures implemented in Java as objects. Arrays store one or more values of a specific data type and provide indexed access to store the same.

- ➢ Arrays must also be declared before use
  - Have fixed size
- ➢ May be specified by a literal, by an expression, or implicitly
  - May be optionally initialized
  - Have default values depending on their type
  - Are always zero-based (array[0] is the first element)



An array of 10 elements.

Types of Arrays are

- Single Dimension Array : is declared to store multiple values in a single variable using single bracket.
- Multidimensional Array : Multidimensional array is declared by using two or more sets of brackets.

Initialization of single dimension array

```
int[] age = {12, 4, 5, 2, 5};
```

Initialization of multidimensional array

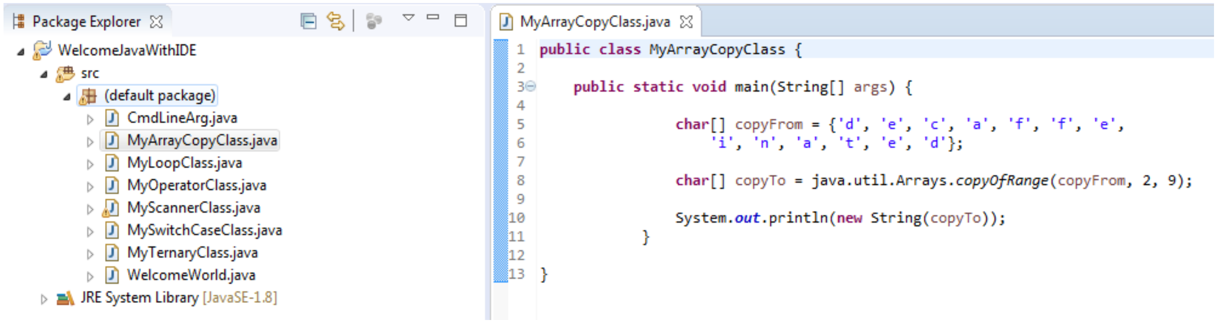
```
Double[][] matrix = { { 1.2, 4.3 }, { 4.1, -1.1 } };
```

Exercise 6: Implement single dimensional array using character

In this exercise you will learn how to implement a character array and use the arraycopy method from java.util package to iterate through elements of the source array and place in destination array.

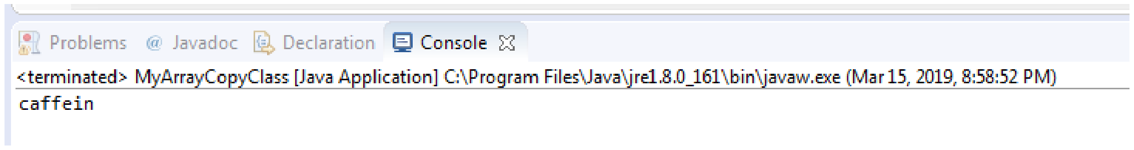
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next.**
4. Enter Project name as **MyArrayCopyClass** and enter the code as shown in the screen capture.



```
public class MyArrayCopyClass {

    public static void main(String[] args) {

            char[] copyFrom = {'d', 'e', 'c', 'a', 'f', 'f', 'e',
                'i', 'n', 'a', 't', 'e', 'd'};

            char[] copyTo = java.util.Arrays.copyOfRange(copyFrom, 2, 9);

            System.out.println(new String(copyTo));
        }
}
```

5. Select **MyArrayCopyClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.



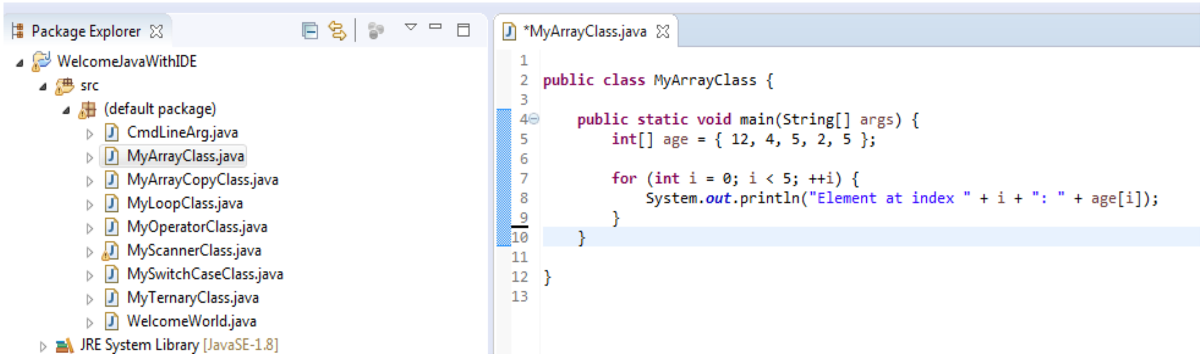
```
<terminated> MyArrayCopyClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 15, 2019, 8:58:52 PM)
caffein
```

Exercise 7: Implement single dimensional array using integer

In this exercise you will learn how to implement an integer array using preceding operator.

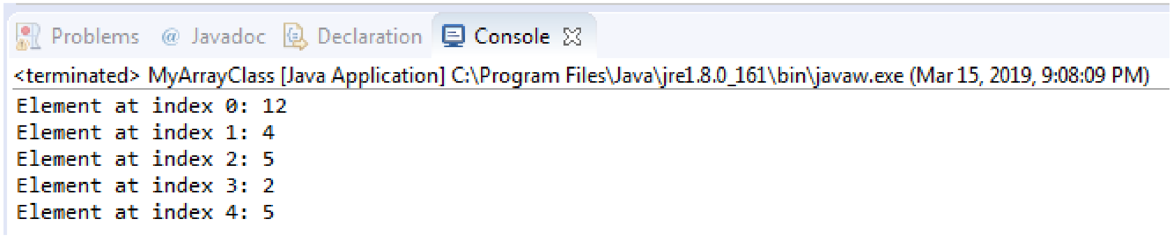
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyArrayClass** and enter the code as shown in the screen capture.



```java

public class MyArrayClass {

    public static void main(String[] args) {
        int[] age = { 12, 4, 5, 2, 5 };

        for (int i = 0; i < 5; ++i) {
            System.out.println("Element at index " + i + ": " + age[i]);
        }
    }

}
```

5. Select **MyArrayClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.



```
<terminated> MyArrayClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 15, 2019, 9:08:09 PM)
Element at index 0: 12
Element at index 1: 4
Element at index 2: 5
Element at index 3: 2
Element at index 4: 5
```

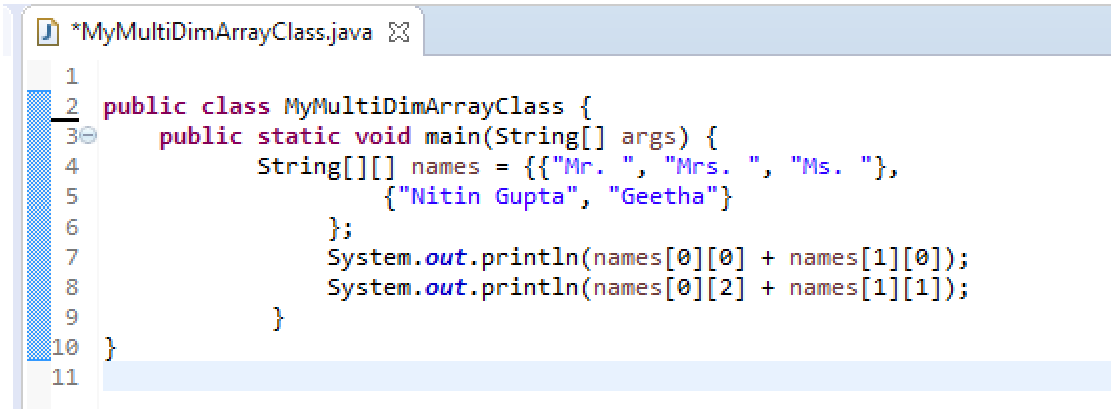
Exercise 8: Implement multidimensional array

In this exercise you will learn how to implement an integer array using preceding operator.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**

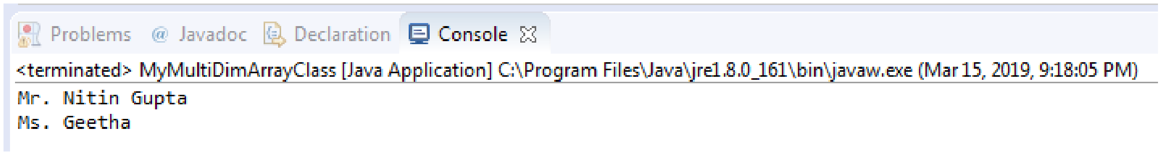
3. Click **Next**.
4. Enter Project name as **MyMultiDimArrayClass** and enter the code as shown in the screen capture.



*MyMultiDimArrayClass.java

```

public class MyMultiDimArrayClass {
    public static void main(String[] args) {
            String[][] names = {{"Mr. ", "Mrs. ", "Ms. "},
                      {"Nitin Gupta", "Geetha"}
                };
                System.out.println(names[0][0] + names[1][0]);
                System.out.println(names[0][2] + names[1][1]);
            }
}

```

5. Select **MyMultiDimArrayClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.



```
<terminated> MyMultiDimArrayClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 15, 2019, 9:18:05 PM)
Mr. Nitin Gupta
Ms. Geetha
```

Test Your Knowledge

1. Which of these is long data type literal?

   A. 0x99fffL
   B. ABCDEFG
   C. 0x99fffa
   D. 99671246

2. Which of these cannot be used for a variable name in Java?

   A. identifier
   B. keyword
   C. identifier & keyword
   D. none of the mentioned

3. Which of these are selection statements in Java?

   A. if()
   B. for()
   C. continue
   D. break

4. Which of the following loops will execute the body of loop even when condition controlling the loop is initially false?

   A. do-while
   B. while
   C. for
   D. none of the mentioned

Java Training Module

5. What is the output of this program?

```
class comma_operator
{
   public static void main(String args[])
   {
       int sum = 0;
       for (int i = 0, j = 0; i < 5 & j < 5; ++i, j = i + 1)
          sum += i;
      System.out.println(sum);
   }
}
```

A. 5
B. 6
C. 14
D. compilation error

Chapter 4: Creating Classes and Objects

Classes and Objects

Classes and Objects are basic concepts of Object-Oriented Programming which revolve around the real-life entities.

- Encapsulate attributes (fields) and behavior (methods)
    - Attributes and behavior are *members* of the class
- Members may belong to either of the following:
    - The whole class (class variables and methods, indicated by the keyword static)

  Individual objects (instance variables and methods)
- Classes can be:
    - Independent of each other
    - Related by inheritance (superclass/subclass)
    - Related by type (interface)

    Syntax

```
    class <class_name>{
        fields;
        method;
    }
```

Implementing classes

Following are the way the classes are implemented:

- Classes are grouped into packages
    - A package contains a collection of logically related classes
- Source code files have the extension .java
    - There is one public class per .java file
- A class is like a blueprint; a class is used to create an object *instance* of the class typically

Class declaration

A class declaration specifies a type

- The identifier in a class declaration specifies the name of the class
- The optional extends clause indicates the superclass

- The optional implements clause lists the names of all the interfaces that the class implements

Class modifiers

- The declaration may include class modifiers (public, abstract, final), which affect how the class can be used
- If the class is declared public, it may be accessed by a Java code that can access its containing package
    - Otherwise, it may be accessed only from within its containing package
- Abstract classes can contain anything that a normal class
    - Cannot be instantiated, only subclass
    - Provide common information for subclasses
- A class is declared final if it permits no subclasses

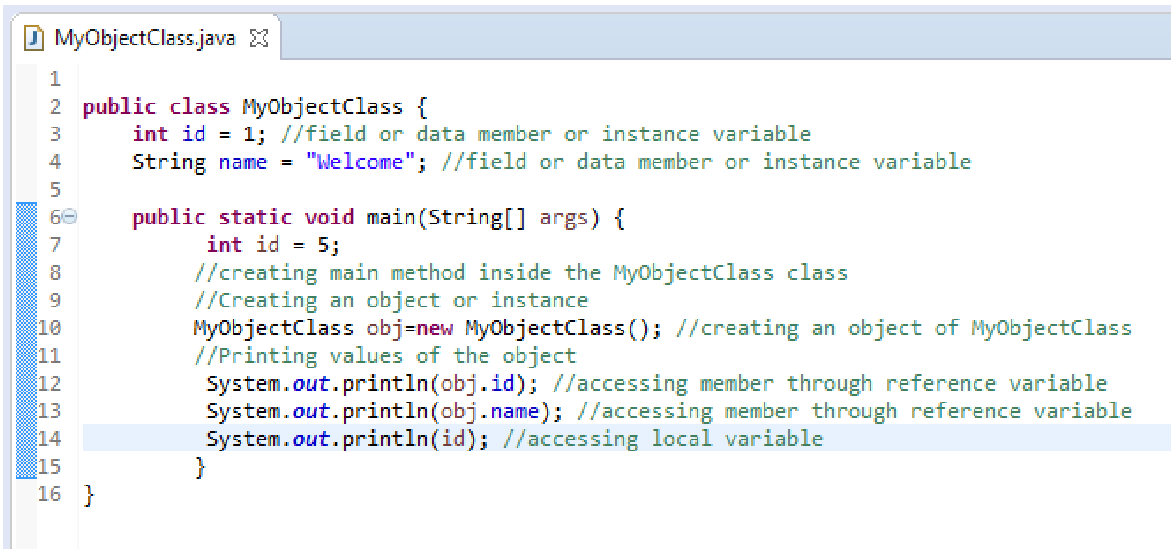
```
_   //Java Program to illustrate how to define a class and fields
    //Defining a Student class.
    class Student{
     //defining fields
     int id;//field or data member or instance variable
     String name;
     //creating main method inside the Student class
     public static void main(String args[]){
      //Creating an object or instance
      Student s1=new Student();//creating an object of Student
      //Printing values of the object
      System.out.println(s1.id);//accessing member through reference
variable
      System.out.println(s1.name);
     }
    }
```

Exercise 1: Implement class and objects

In this exercise you will learn how create an object for a class.

Procedure

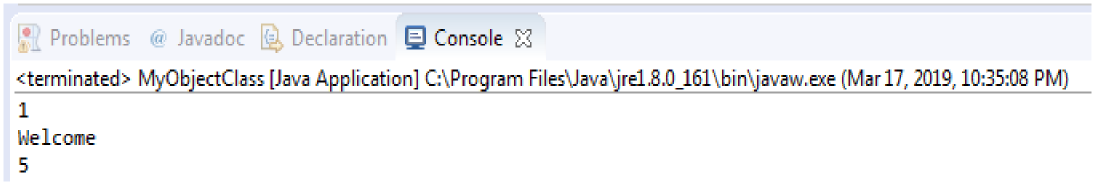
1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyObjectClass** and enter the code as shown in the screen capture.



MyObjectClass.java

```java

public class MyObjectClass {
    int id = 1; //field or data member or instance variable
    String name = "Welcome"; //field or data member or instance variable

    public static void main(String[] args) {
          int id = 5;
         //creating main method inside the MyObjectClass class
         //Creating an object or instance
         MyObjectClass obj=new MyObjectClass(); //creating an object of MyObjectClass
         //Printing values of the object
          System.out.println(obj.id); //accessing member through reference variable
          System.out.println(obj.name); //accessing member through reference variable
          System.out.println(id); //accessing local variable
         }
}
```

5. Select **MyObjectClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.



```
<terminated> MyObjectClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 17, 2019, 10:35:08 PM)
1
Welcome
5
```

Java Training Module

Methods

- Methods define how an object responds to messages
- Methods define the behavior of the class
  - o All methods belong to a class

Method signatures

- A class can have many methods with the same name
  - o Each method must have a different signature
- The method signature consists of:
  - The method names
  - Argument number and types

```
public static int methodName(int a, int b) {
   // body
}
```

Method parameters

- Arguments (parameters) are passed:
  - o by value for primitive types
  - o by object reference for reference types
- Primitive values cannot be modified when passed as an argument

```
public void method1(){
        int a=0;
        System.out.println(a);
outputs 0
     method2(a);
     System.out.println(a);
outputs 0
}
```

```
void method2(int a){
        a=a+1;
}
```

Returning from methods

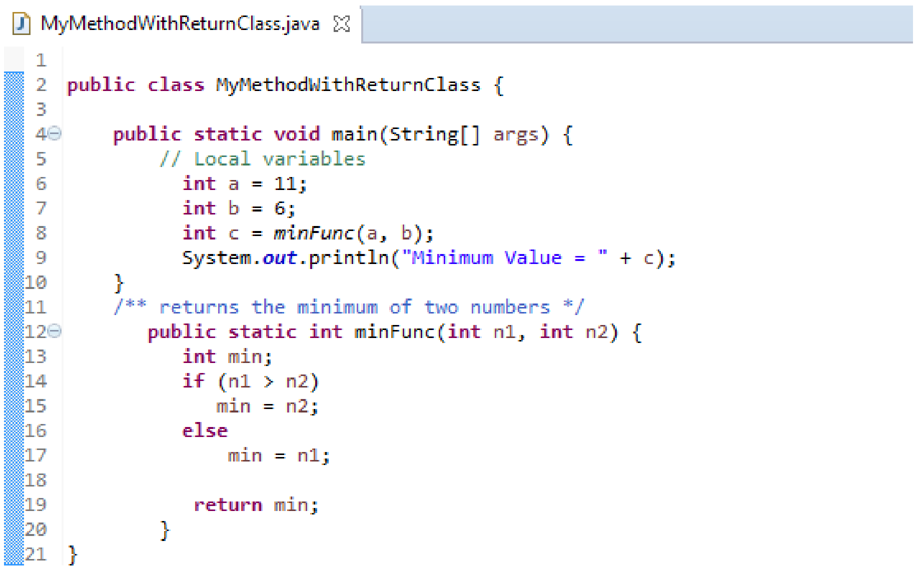
- Methods return, at most, one value or one object
    - o If the return type is void, the return statement is optional
- The return keyword is used to return control to the calling method.
- There may be several return statements in a method; the first one reached is executed

Exercise 2: Implement methods with return type

In this exercise you will learn how create an object for a class.

Procedure

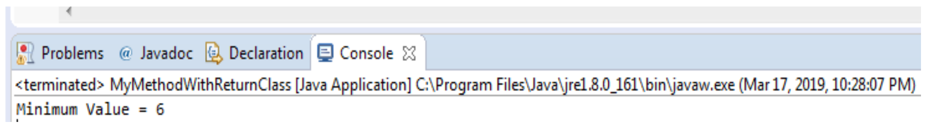
1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyMethodWithReturnClass** and enter the code as shown in the screen capture.



MyMethodWithReturnClass.java

```

public class MyMethodWithReturnClass {

    public static void main(String[] args) {
        // Local variables
          int a = 11;
          int b = 6;
          int c = minFunc(a, b);
          System.out.println("Minimum Value = " + c);
    }
    /** returns the minimum of two numbers */
       public static int minFunc(int n1, int n2) {
          int min;
          if (n1 > n2)
             min = n2;
          else
              min = n1;

           return min;
        }
}
```

5. Select **MyMethodWithReturnClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.



```
<terminated> MyMethodWithReturnClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 17, 2019, 10:28:07 PM)
Minimum Value = 6
```

Invoking methods

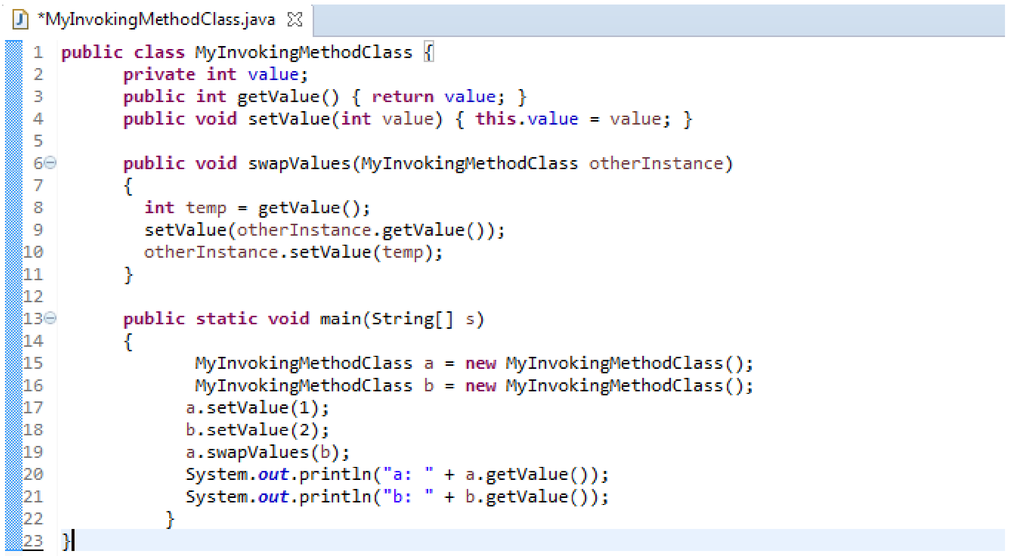
To call a method, use the dot (.) operator. The same operator is used to call both class and instance methods. If the call is to a method of the same class, the dot operator is not necessary.

Exercise 3: Implement methods with return type

In this exercise you will learn how to invoke method to swap the integer values using class objects.

Procedure

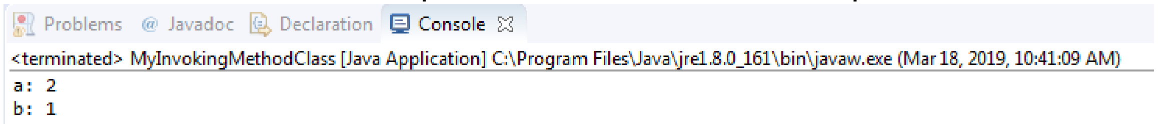
1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyInvokingMethodClass** and enter the code as shown in the screen capture.



*MyInvokingMethodClass.java

```
public class MyInvokingMethodClass {
      private int value;
      public int getValue() { return value; }
      public void setValue(int value) { this.value = value; }

      public void swapValues(MyInvokingMethodClass otherInstance)
      {
        int temp = getValue();
        setValue(otherInstance.getValue());
        otherInstance.setValue(temp);
      }

      public static void main(String[] s)
      {
             MyInvokingMethodClass a = new MyInvokingMethodClass();
             MyInvokingMethodClass b = new MyInvokingMethodClass();
            a.setValue(1);
            b.setValue(2);
            a.swapValues(b);
            System.out.println("a: " + a.getValue());
            System.out.println("b: " + b.getValue());
        }
}
```

5. Select **MyInvokingMethodClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.



```
<terminated> MyInvokingMethodClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 18, 2019, 10:41:09 AM)
a: 2
b: 1
```

Constructors

The class body contains at least one *constructor*, which is a method that sets up a new instance of a class

- The method has the same name as the class
- Use the new keyword with a constructor to create *instances* of a class

Memory management in Java

- Since Java does not use pointers, memory addresses cannot be accidentally or maliciously overwritten
- The problems inherent in user allocated and deallocated memory are avoided, since the Java Virtual
  Machine handles all memory management
- Programmers do not have to keep track of the memory the allocate from the heap and explicitly deallocate it

More about constructors

- Constructors are used to create and initialize objects
    - o A constructor always has the same name as the class it constructs (case-sensitive)
- Constructors have no return type
    - o Constructors return no value, but when used with new, return a reference to the new object

Default constructors

- The constructor with no arguments is a default constructor
- The Java platform provides a default constructor only if you do not explicitly define any constructor
- When defining a constructor, you should also provide a default constructor

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Text for this section should be provided here.

This Keyword

this is a keyword in Java which is used as a reference to the object of the current class, with in an instance method or a constructor. Using *this* you can refer the members of a class such as constructors, variables and methods.

The keyword this is used only within instance methods or constructors.

Java Training Module

In general, the keyword *this* is used to –

- Differentiate the instance variables from local variables if they have same names, within a constructor or a method.

```
class Student {
   int age
   Student() {
      this(20);
   }

   Student(int age) {
      this.age = age;
   }
}
```

- Call one type of constructor (parametrized constructor or default) from other in a class. It is known as explicit constructor invocation.

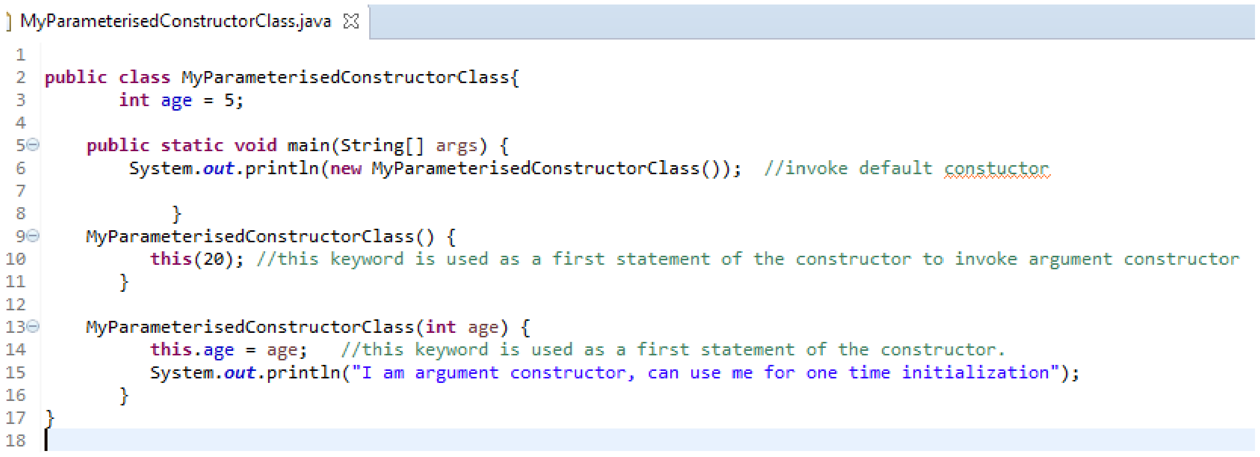
```
class Student {
   int age
   Student() {
      this(20);
   }

   Student(int age) {
      this.age = age;

   }
}
```

Exercise 4: Implement constructors

In this exercise you will learn how to create constructors and use this keyword to invoke constructors.

Procedure

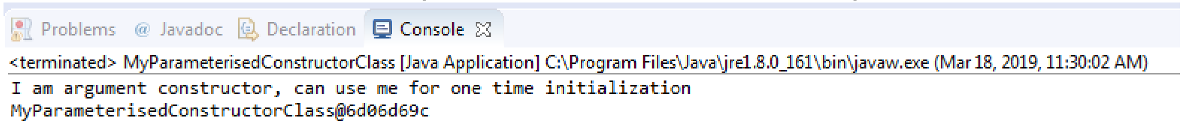
1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyParameterisedConstructorClass** and enter the code as shown in the screen capture.



MyParameterisedConstructorClass.java

```

public class MyParameterisedConstructorClass{
        int age = 5;

     public static void main(String[] args) {
         System.out.println(new MyParameterisedConstructorClass());  //invoke default constuctor

             }
     MyParameterisedConstructorClass() {
           this(20); //this keyword is used as a first statement of the constructor to invoke argument constructor
        }

     MyParameterisedConstructorClass(int age) {
           this.age = age;   //this keyword is used as a first statement of the constructor.
           System.out.println("I am argument constructor, can use me for one time initialization");
        }
}

```

5. Select **MyParameterisedConstructorClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.



```
<terminated> MyParameterisedConstructorClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 18, 2019, 11:30:02 AM)
I am argument constructor, can use me for one time initialization
MyParameterisedConstructorClass@6d06d69c
```

Overloading

Method Overloading

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we must perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading increases the readability of the program.

Different ways to overload the method

There are two ways to overload the method in java

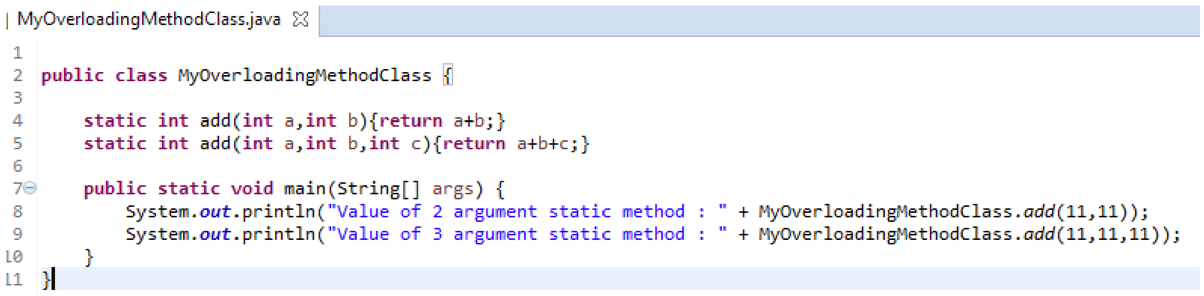
- By changing number of arguments
- By changing the data type

Exercise 5: Implement overloading static methods

In this exercise you will create two methods, first add method performs addition of two numbers and second add method performs addition of three numbers. Also, will create static methods so that we don't need to create instance for calling methods.

Procedure

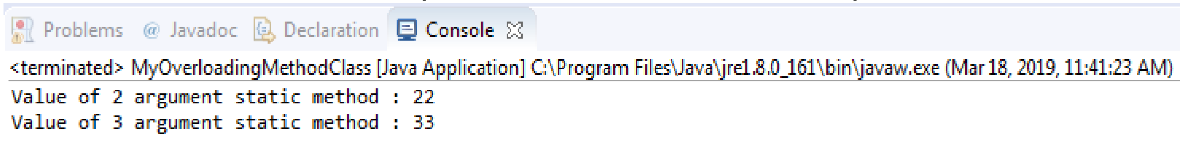
1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyOverloadingMethodClass** and enter the code as shown in the screen capture.



MyOverloadingMethodClass.java

```

public class MyOverloadingMethodClass {

    static int add(int a,int b){return a+b;}
    static int add(int a,int b,int c){return a+b+c;}

    public static void main(String[] args) {
        System.out.println("Value of 2 argument static method : " + MyOverloadingMethodClass.add(11,11));
        System.out.println("Value of 3 argument static method : " + MyOverloadingMethodClass.add(11,11,11));
    }
}
```

5. Select **MyOverloadingMethodClass.java** from Package Explorer.
6. Click **Run** to view the output as shown in screen capture.



```
<terminated> MyOverloadingMethodClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 18, 2019, 11:41:23 AM)
Value of 2 argument static method : 22
Value of 3 argument static method : 33
```

Constructor Overloading

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Exercise 6: Implement overloading static methods

In this exercise you will create two methods, first add method performs addition of two numbers and second add method performs addition of three numbers. Also, will create static methods so that we don't need to create instance for calling methods.

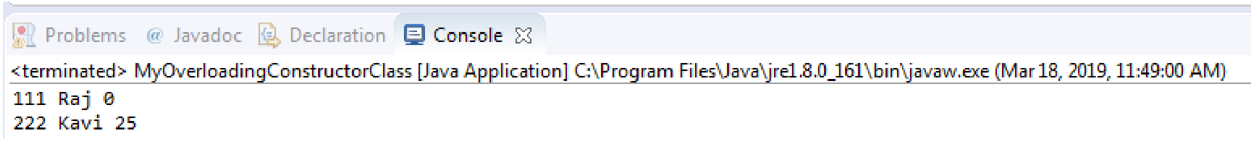
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyOverloadingConstructorClass** and enter the code as shown in the screen capture.

```
//Java program to overload constructors in java
    class MyOverloadingConstructorClass {
        int id;
        String name;
        int age;
        //creating two arg constructor
        MyOverloadingConstructorClass(int i,String n){
        id = i;
        name = n;
        }
        //creating three arg constructor
        MyOverloadingConstructorClass(int i,String n,int a){
        id = i;
        name = n;
        age=a;
        }
        void display(){System.out.println(id+" "+name+" "+age);}

        public static void main(String args[]){
        MyOverloadingConstructorClass s1 = new MyOverloadingConstructorClass
(111,"Raj");
        MyOverloadingConstructorClass s2 = new MyOverloadingConstructorClass
(222,"Kavi",25);
        s1.display();
        s2.display();
       }
    }
```

5. Select **MyOverloadingMethodClass.java** from Package Explorer.

6. Click **Run** to view the output as shown in screen capture.



```
<terminated> MyOverloadingConstructorClass [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 18, 2019, 11:49:00 AM)
111 Raj 0
222 Kavi 25
```

Static members and Initialization Blocks

Static Class members

In Java you can create a field or method that does not belong to any object of a class. Such are known as static variables or static methods.

Static Variable

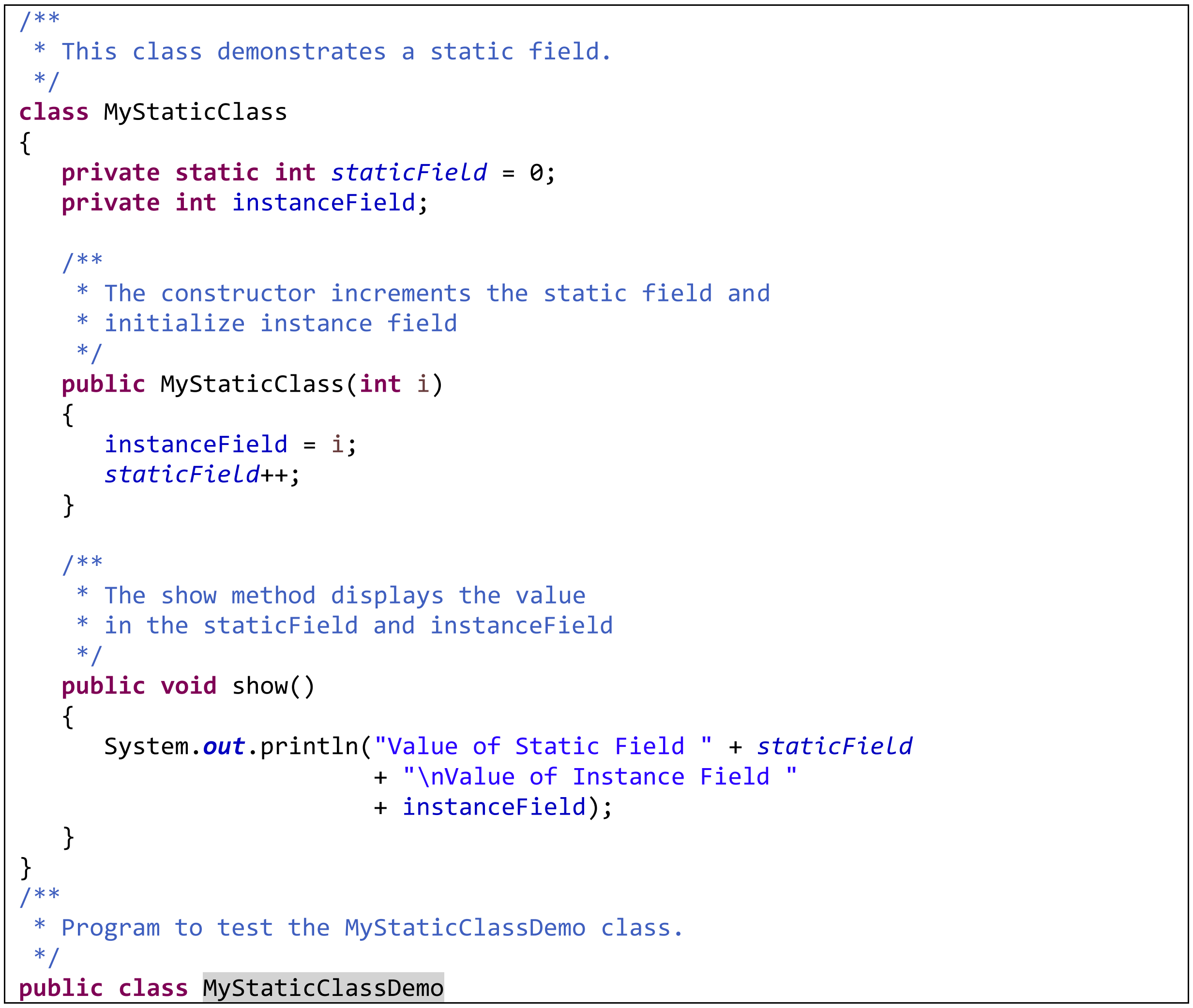
To declare a static variable, you put them in the class just like instance variables, but you put the keyword static in front. Static variables belong to the class and has one copy. When a value is stored in a static variable, it is not stored in object of the class. Static variables are useful to keep information that is common to every object.

Exercise 7: Implement static variable

In this exercise you will understand the implementation of static and non-static variable.

Procedure

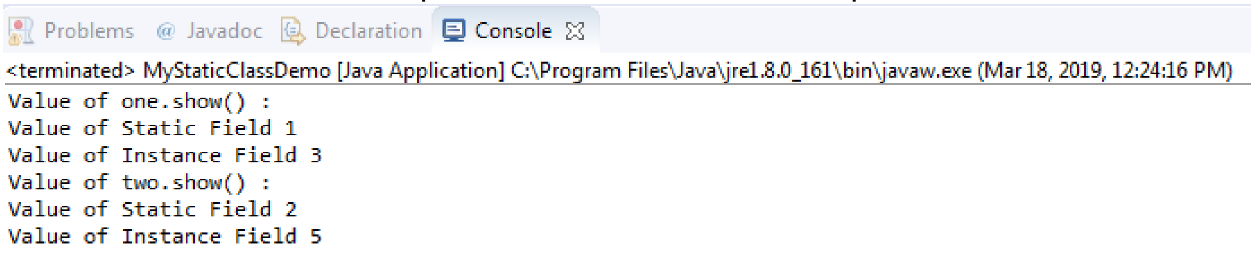
1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyOverloadingConstructorClass** and enter the code as shown in the screen capture.



```
/**
 * This class demonstrates a static field.
 */
class MyStaticClass
{
   private static int staticField = 0;
   private int instanceField;

   /**
    * The constructor increments the static field and
    * initialize instance field
    */
   public MyStaticClass(int i)
   {
      instanceField = i;
      staticField++;
   }

   /**
    * The show method displays the value
    * in the staticField and instanceField
    */
   public void show()
   {
      System.out.println("Value of Static Field " + staticField
                         + "\nValue of Instance Field "
                         + instanceField);
   }
}
/**
 * Program to test the MyStaticClassDemo class.
 */
public class MyStaticClassDemo
```

```
{
   public static void main(String[] args)
   {
         MyStaticClass one = new MyStaticClass(3);

      System.out.println("Value of one.show() : ");
      one.show();

      MyStaticClass two = new MyStaticClass(5);

      System.out.println("Value of two.show() : ");
      two.show();
   }
}
```

5. Click **Run** to view the output as shown in screen capture.


Problems @ Javadoc Declaration Console

```
<terminated> MyStaticClassDemo [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Mar 18, 2019, 12:24:16 PM)
Value of one.show() :
Value of Static Field 1
Value of Instance Field 3
Value of two.show() :
Value of Static Field 2
Value of Instance Field 5
```

Static Methods

A static method can be accessed by using the name of the class. So, you don't need to create an instance to access the method

Exercise 8: Implement static method

In this exercise you will understand the implementation of static and non-static method.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next.**
4. Enter Project name as **MyMathDemo** and enter the code as shown in the screen capture.

```
/**
 * This class demonstrates static methods.
 */
class MyMath
{
   /**
    * The cube method returns the cube of
    * the number passed into parameter.
    */
   public static int cube(int number)
   {
      return number * number * number;
   }
}
/**
 * This program demonstrates the MyMath class.
 */
public class MyMathDemo
{
   public static void main(String[] args)
   {
      // Display the cube of number.
      System.out.println("cube of 3 is " + MyMath.cube(3));
   }
}
```

6. Click **Run** to view the output.

```
cube of 3 is 27
```

Instance Initialization Block

In a Java program, operations can be performed on methods, constructors and initialization blocks. Instance Initialization Blocks or IIB are used to initialize instance variables. IIBs are executed before constructors. They run each time when object of the class is created.

- Initialization blocks are executed whenever the class is initialized and before constructors are invoked.
- They are typically placed above the constructors within braces.
- It is not at all necessary to include them in your classes.

Garbage Collection

Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

Garbage Collectable Objects

The garbage collector sweeps through the JVM's list of objects periodically and reclaims the resources held by unreferenced objects

- All objects that have no object references are eligible for garbage collection
  - References out of scope, objects to which you have assigned null, and so forth
- The JVM decides when the garbage collector is run
  - Typically, the garbage collector is run when memory is low
  - May not be run at all
  - Unpredictable timing

Working with the garbage collector

- You cannot prevent the garbage collector from running, but you can request it to run soon
    - System.gc()
    - This is only a request, not a guarantee
- The finalize() method of an object is run immediately before garbage collection occurs
    - This method should only be used for special cases (such as cleaning up memory allocation from native calls) because of the unpredictability of the garbage collector
- Things like open sockets, files, and so forth should be cleaned up during normal program flow before the object is dereferenced.

Exercise 9: Implement garbage collection

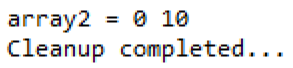
In this exercise you will understand the implementation of deallocation of memory and steps to add java class into packages.

Procedure

5. From the project you have created in the eclipse IDE tool.
6. Click **File** > **Project** > **Java** > **Java Project**
7. Click **Next**.
8. Enter Project name as **MySystemDemo** and enter the code as shown in the screen capture.

```
public class MySystemDemo{

   public static void main(String[] args) {

      int arr1[] = { 0, 1, 2, 3, 4, 5 };
      int arr2[] = { 0, 10, 20, 30, 40, 50 };

      // copies an array from the specified source array
      System.arraycopy(arr1, 0, arr2, 0, 1);
      System.out.print("array2 = ");
      System.out.print(arr2[0] + " ");
      System.out.println(arr2[1] + " ");

      // it runs the GarbageCollector
      System.gc();
      System.out.println("Cleanup completed...");
}
}
```

7. Click **Run** to view the output.



```
array2 = 0 10
Cleanup completed...
```

Packages

- Classes can be grouped:
  - Logically, according to the model you are building
  - As sets designed to be used together
  - For convenience
- By convention, package names are in lower case
- Different packages can contain classes with the same name

I have created a class `AddTwo` inside a package name `AddNumbers`. To create a class inside a package, declare the package name in the first statement in your program. A class can have only one package declaration.

Class visibility

- Classes can reference other classes within the same package by class name only
- Classes must provide the fully qualified name (including package) for classes defined in a different package.

Import statement

- Use import statements to import packages or classes to make other classes directly visible to your class.

Access control

Access level modifiers determine whether other classes can use a field or invoke a method. There are two levels of access control:

- At the top level—public, or *package-private* (no explicit modifier).
- At the member level—public, private, protected, or *package-private* (no explicit modifier).

A class may be declared with the modifier public, in which case that class is visible to all classes everywhere. If a class has no modifier (the default, also known as *package-private*), it is visible only within its own package.

At the member level, you can also use the public modifier or no modifier (*package-private*) just as with top-level classes, and with the same meaning. For members, there are two additional access modifiers: private and protected. The private modifier specifies that the member can only be accessed in its own class. The protected modifier specifies that the member can only be accessed within its own package (as with *package-private*) and, in addition, by a subclass of its class in another package.

Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

The first data column indicates whether the class itself has access to the member defined by the access level. As you can see, a class always has access to its own members. The second column indicates whether classes in the same package as the class (regardless of their parentage) have access to the member. The third column indicates whether subclasses of the class declared outside this package have access to the member. The fourth column indicates whether all classes have access to the member.

Access levels affect you in two ways. First, when you use classes that come from another source, such as the classes in the Java platform, access levels determine which members of those classes your own classes can use. Second, when you write a class, you need to decide what access level every member variable and every method in your class should have.

Let's look at a collection of classes and see how access levels affect visibility. The following figure shows the four classes in this example and how they are related.

Tips on Choosing an Access Level

If other programmers use your class, you want to ensure that errors from misuse cannot happen. Access levels can help you do this.

- Use the most restrictive access level that makes sense for a member. Use private unless you have a good reason not to.

Avoid public fields except for constants. (Many of the examples in the tutorial use public fields. This may help to illustrate some points concisely but is not recommended for production code.) Public fields tend to link you to an implementation and limit your flexibility in changing your code.

Exercise 10: Implement package and import

In this exercise you will understand the implementation of package and import keyword.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **AddTwoNumClass** and

```
package com.addnumber;

public class AddTwoNumClass{

	public static void main(String[] args) {
		 AddTwoNumClass obj = new AddTwoNumClass();
		 System.out.println(obj.add(10, 20));
		 }
	public int add(int a, int b){
		return a+b;
	    }
}
```

Now let's see how to use this package in another program.

```
import com.addnumber.AddTwoNumClass;

public class MyAddDemo{

	public static void main(String[] args) {
		AddTwoNumClass obj = new AddTwoNumClass();
		System.out.println(obj.add(100, 200));
	}
}
```

5. Click **Run** to view the output.

```
300
```

Test Your Knowledge

1. Which of the following is not type of class?

A. Abstract Class
B. Final Class
C. Start Class
D. String Class

2. Which among the following is not a necessary condition for constructors?

A. Its name must be same as that of class
B. It must not have any return type
C. It must contain a definition body
D. It can contain arguments

3. Default constructor must be defined, if parameterized constructor is defined and the object is to be created without arguments.

A. True
B. False

4. If class C inherits class B. And B has inherited class A. Then while creating the object of class C, what will be the sequence of constructors getting called?

A. Constructor of C then B, finally of A
B. Constructor of A then C, finally of B
C. Constructor of C then A, finally B
D. Constructor of A then B, finally C

Java Training Module

5. Which of the following is incorrect?

   A. class student{ }s;
   B. class student{ }; student s;
   C. class student{ }s[];
   D. class student{ }; student s[5];

Chapter 5: Implementing OOP Concepts

Implementing Encapsulation

By providing only a setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.

It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.

The encapsulate class is easy to test. So, it is better for unit testing.

The standard IDE's are providing the facility to generate the getters and setters. So, it is easy and fast to create an encapsulated class in Java.

Implementing Aggregation

Any class that have an entity reference, it is known as Aggregation. It represents HAS-A relationship. It is used for Code Reusability.

Exercise 1: Implement Encapsulation

In this exercise you will understand the implementation of encapsulation to hide the data members of the class.

Procedure

6. From the project you have created in the eclipse IDE tool.
7. Click **File** > **Project** > **Java** > **Java Project**
8. Click **Next**.
9. Project name as **BankAccount** and store in a new package **com.megabank.model**.

```
package com.megabank.model;

public class BankAccount {
	private String owner;
	private double balance = 0;
	public String getOwner(){
		return owner;
	}
	public void setOwner(String name){
		owner = name;
	}
	public double getBalance(){
		return balance;
	}
	public void setBalance(double amt){
		balance = amt;
	}
}
```

10. Now let’s see create another class **MyBankAccountDemo**, how to access the data variable through methods in another program.

```
import com.megabank.model;

public class MyBankAccountDemo{

	public static void main(String[] args) {
		BankAccount obj = new BankAccount();
		obj.setOwner(“IBM Corporation”);
		obj.setBalance(“9999999.999”);
		System.out.println(obj.getOwner() +”  “+obj.setBalance());
	}
}
```

11. Click **Run** to view the output.

```
IBM Corporation = 9999999.999
```

Implementing Aggregation

Exercise 2: Implement Aggregation

In this exercise you will understand the implementation of aggregation to showcase multiple functionality.

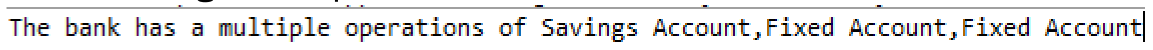
The class name and the file name are different; hence you should run the Operation.class to view the result.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyOperationClass**.

```
class Operation{
             //class operation has multiple functionality to the customers. Each
function has different process.
             String savingsAccount(){
                    return "Savings Account";
             }
             String fixedAccount(){
                    return "Fixed Account";
             }
             String currentAccount(){
                    return "Current Account";
             }
             public static void main(String[] args) {
                    Operation obj = new Operation();
               System.out.print("The bank has a multiple operation of " +
obj.savingsAccount()+ ","+ obj.fixedAccount() +","+ obj.fixedAccount());
             }
}
```

5. Click **Run** against Operation class to view.



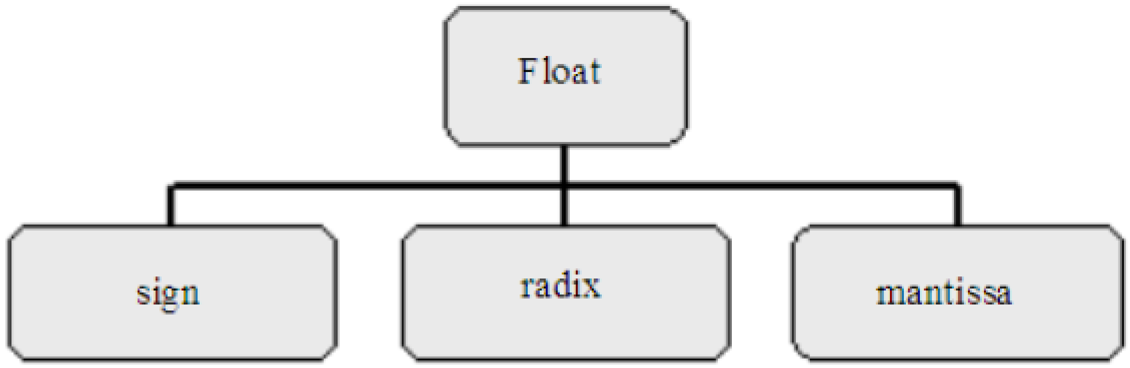
```
The bank has a multiple operations of Savings Account,Fixed Account,Fixed Account
```

Inheritance

Overview of inheritance and IS-A relationship

Inheritance describes relationships in which an instance of the subclass. Subclass is a special case of the superclass inheritance relationships are called is a relationship because the subclass is a special case of the superclass.

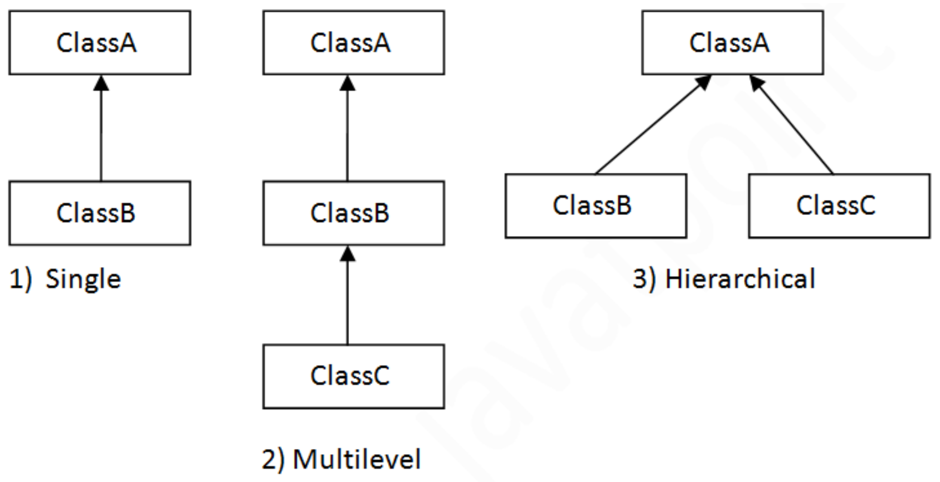
Example: A floating point number has a sign; a radix and a mantissa and collections provide generalized ways to handle has-a relationships.



Single vs Multiple level inheritance

Inheritance represents the IS-A relationship. These relationships are also called as

Sub Class/Child Class	Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class
Super Class/Parent Class	Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
Reusability	Facilitates you to reuse the fields and methods of the existing class when you create a new class.



Method overriding

Method overriding is used to provide the specific implementation of a method which is already provided by its superclass. Method overriding is used for runtime polymorphism.
The overriding method must have

- the same name as in the parent class
- the same parameter as in the parent class.
- IS-A relationship (inheritance).

Restriction of overriding method

Restrictions on the overriding method:

- The parameter list must match the inherited method exactly.
- The return type must be the same as that of the inherited method.
- The access modifier must not be more restrictive than that of for example, if overriding a protected method, the new method can be protected or public, but not private.

Exercise 3: Implement Inheritance

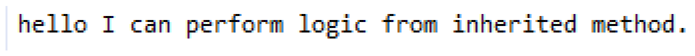
In this exercise you will understand the concepts of inheritance.

Procedure

6. From the project you have created in the eclipse IDE tool.
7. Click **File** > **Project** > **Java** > **Java Project**
8. Click **Next**.
9. Enter Project name as **MySuperClass**.

```
class MySuperClass {
        static String name = "  test super  ";
                public static String superTest(String s){
                        s += " I can perform logic from inherited method. ";
                        return s;
                }
}

class MySubClass extends MySuperClass{
        public static void main(String []args){
                System.out.println(superTest("hello"));
        }
}
```

10. Expand **MySuperClass** in **Package Explorer**, RightClick **MySubClass** > **Run** > **Java Application** to view the output.



```
hello I can perform logic from inherited method.
```

Cosmic class

The Object class is the ultimate ancestor. Every class in Java extends Object. This Object class is called as Cosmic class.

The ultimate superclass Object is taken for granted if no superclass is explicitly mentioned. Because every class in Java extends Object, it is important to be familiar with the services provided by the Object class.

Exercise 4: Implement Inheritance

In this exercise you will understand the concepts of inheritance.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyObjectEqualsClass**.

```
public class MyObjectEqualsClass{
        public static void main(String[] args) {
                String name = "Joe";
                if("Joe".equals(name))
                        name += " Smith";
                System.out.println(name); // Validates and prints Joe only if name is
Joe.
             name ="JOHN";
         if("John".equalsIgnoreCase(name))
                 name += " Smith";
          System.out.println(name); // Validates and prints John irrespective of
case.
        }
}
```

5. Click **Run** to view the output.

```
Joe Smith
JOHN Smith
```

String Vs StringBuffer

StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

Exercise 5: Implement String

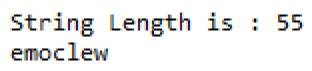
In this exercise you will understand the concepts of String and StringBuffer class.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Enter Project name as **MyObjectEqualsClass**.

```
public class MyStringClass{

  public static void main(String args[]) {
    String palindrome = "I observed a brilliant innovative idea in
this student.";
    int len = palindrome.length();
    System.out.println( "String Length is : " + len );

    StringBuffer sb = new StringBuffer("welcome");
    System.out.println(sb.reverse());
    }
}
```

5. Click **Run** to view the output.



```
String Length is : 55
emoclew
```

Polymorphism

A compact car, luxury car, and sports car are several classes of car that accept the same set of messages, and provide the same service

Each car accepts the

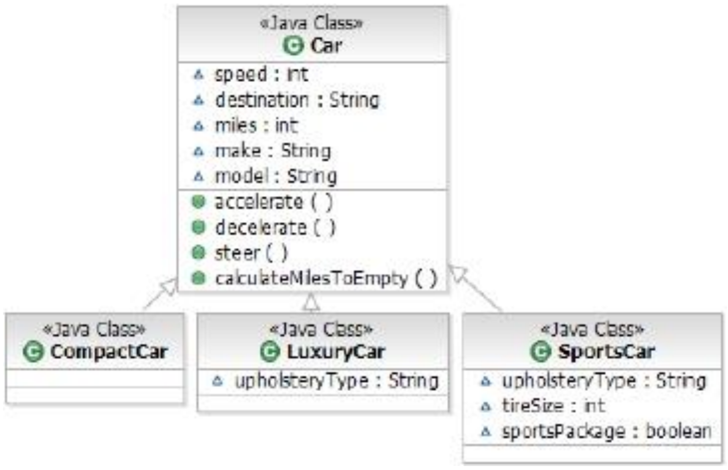
- accelerate()
- decelerate()
- steer() and
- calculateMilesToEmpty()

messages, allowing you to drive to a destination

- The service may be implemented differently by each car, but these classes may be interchanged without affecting the driver who sends messages to the vehicle. This principle is known as polymorphism.
- Classes inherit from the same superclass inherit the same methods and can respond to the same messages.
- CompactCar, LuxuryCar and SportsCar can respond to the same messages, and be interchanged without affecting the message sender, a driver.
- Inheritance is one way that Java implements polymorphism. Check the inheritance relationship between classes that model these cars.

Implementing polymorphism

Assigning an object of one type to an object of another type (higher in the hierarchy), makes the object forget its real type. For Example,



```
Car auto = new Car();
Car auto = new CompactCar();
Car auto = new LuxuryCar();
Car auto = new SportsCar();
```

Any class that have an entity reference, it is known as Aggregation. It represents HAS-A relationship. It is used for Code Reusability.

Upcasting and Down casting

Upcasting means casting the object to a superclass, while downcasting means casting to a subclass. Upcasting is not necessary as it's done automatically. And it's usually referred as implicit casting.

For Example, CompactCar and only responds to messages for the Car class. You can get these objects to remember their real type by casting the object to that type is called as upcasting.

```
CompactCar cc = (CompactCar)auto
```

Casting to an unrecognized subclass throws a **ClassCastException**.

Exercise 6: Implement Polymorphism

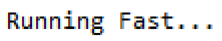
In this exercise you will understand the implementation of polymorphism through upcasting.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyPolymorphismClass**.

```
class Person{
        void walk(){
                System.out.println("Can Run....");
        }
}
class Employee extends Person{
        void walk(){
                System.out.println("Running Fast...");
        }
        public static void main(String arg[]){
                Person p=new Employee(); //upcasting
                p.walk();
        }
}
```

5. Click **Run** to view the output.



```
Running Fast...
```

Abstract Classes and Interfaces

Abstract

Abstract keyword is used to create an abstract class and method.

- Abstract classes cannot be instantiated; they are intended to be a superclass for other classes
- Abstract methods have no implementation
- If a class has one or more abstract methods, it is abstract, and must be declared so
- Concrete classes have full implementations and can be instantiated.

Interfaces

Many classes of vehicle provide the same services as the car class: trucks, boats, planes, and so on. These vehicles accept the same set of messages that allow them to be driven, and thus have the same interface to their services, such as:

- Accelerate
- Decelerate
- Steer

Interfaces provide a list of methods separate from the actual implementation of these methods.

Abstract classes vs Interfaces

The difference between abstract classes and interfaces with respect to its members are:

Members	Abstract classes	Interfaces
Methods	can have abstract and non-abstract methods including default and static methods.	can have only abstract methods.

Final Variables	may contain non-final variables.	variables declared in a Java interface are by default final.
Type of variables	can have final, non-final, static and non-static variables.	has only static and final variables.
Implementation	can provide the implementation of interface.	can't provide the implementation of abstract class.
Inheritance vs Abstraction	can be extended using keyword "extends".	can be implemented using keyword "implements".
Multiple implementation	can extend another Java class and implement multiple Java interfaces.	can extend another Java interface only.
Accessibility of Data Members	can have class members like private, protected, etc.	Members of a Java interface are public by default.

Loose coupling

In object-oriented design, coupling refers to the degree of direct knowledge that one element has of another. Like class A force related changes in class B.

There are two types of coupling:

Tight coupling

Tight coupling means the two classes often change together. Means, if class A knows more than it should about the way in which class B was implemented, then A and B are tightly coupled.

Loose coupling

Loose coupling means they are mostly independent. Means, if class A has about class B, is what class B has exposed through its interface, then class A and class B are said to be loosely coupled.

In order to overcome from the problems of tight coupling between objects, spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection it's possible to achieve loose coupling.

Exercise 7: Implement interface

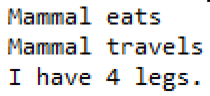
In this exercise you will understand the implementation of single inheritance and multilevel inheritance through abstract class and interface.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyInterfaceClass**.

```
// All types of have common features to eat, travel whereas the method
to perform is different
interface Animal {
  public void eats();
  public void travel();
}
class Mammal implements Animal {
  public void eats() {
    System.out.println("Mammal eats");
  }
  public void travel() {
    System.out.println("Mammal travels");
  }
  public int noOfLegs() {
    return 4;
  }
  public static void main(String args[]) {
    Mammal m = new Mammal();
    m.eats();
    m.travel();
  System.out.println("I have " +m.noOfLegs() +" legs.");   }}
```

5. Expand **MyInterfaceClass** in Package Explorer, Select **Mammal** > **Run As** > **Java Application** to view the output.



```
Mammal eats
Mammal travels
I have 4 legs.
```

Exercise 8: Implement abstract

In this exercise you will understand the implementation of abstract class.

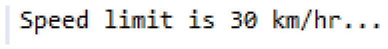
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyAbstractClass**.

```
abstract class Vehicle{
        abstract void speed();  // abstract method
}
class Bike extends Vehicle{
        void speed(){
                System.out.println("Speed limit is 60 km/hr..");
        }
        public static void main(String args[]){
                Vehicle obj = new Bike(); //indirect object creation
                obj.speed();
        }
}
class Moped extends Vehicle{
        void speed(){
                System.out.println("Speed limit is 30 km/hr...");
        }
        public static void main(String args[]){
                Vehicle obj = new Moped(); //indirect object creation
                obj.speed();
        }
}
```

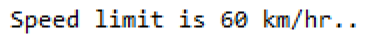
Java Training Module

5. Expand **MyAbstractClass** in Package Explorer, select **Moped** > **Run As** > **Java Application** to view the output.



```
Speed limit is 30 km/hr...
```

6. Expand **MyAbstractClass** in Package Explorer, select **Bike** > **Run As** > **Java Application** to view the output.



```
Speed limit is 60 km/hr..
```

Test Your Knowledge

1. If a class inheriting an abstract class does not define all of its function then it will be known as?

   A. Abstract
   B. A simple class
   C. Static class
   D. None of the mentioned

2. What is the output of this program?

```
class A  {
          public int i;
          public int j;
          A() {
             i = 1;
             j = 2;
         }
}
 class B extends A  {
          int a;
         B() {
             super();
          }
 }
 class super_use {
          public static void main(String args[]){
             B obj = new B();
             System.out.println(obj.i + " " + obj.j)
          }
      }
```

A. 1 2
B. 2 1
C. Runtime Error
D. Compilation Error

3. Which among the following can’t be used for polymorphism?

A. Static member functions
B. Member functions overloading
C. Predefined operator overloading
D. Constructor overloading

4. Which among the following can show polymorphism?

A. Overloading ||
B. Overloading +=
C. Overloading <<
D. Overloading &&

5. If a class inheriting an abstract class does not define all of its function then it will be known as?

A. Abstract
B. A simple class
C. Static class
D. None of the mentioned

Chapter 6:Useful JAVA API Classes

Working with strings

String Class

The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings.

String concatenation is implemented through the StringBuilder(or StringBuffer) class and it append method.

String conversions are implemented through the method toString, defined by Object and inherited by all classes in Java.

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as

- concatenating
- converting
- trimming
- comparing
- replacing strings

Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

StringBuilder Class

StringBuilder is non-synchronized. It means two threads can call the methods of StringBuilder simultaneously.
The features of the StringBuilder class are:

- A mutable sequence of characters. This class provides an API compatible with StringBuffer, but with no guarantee of synchronization.
- This class is designed for use as a drop-in replacement for StringBuffer in places where the string buffer was being used by a single thread.
- This class is used in preference to StringBuffer as it will be faster under most implementations.

The principal methods on a StringBuilder are

- append() : Adds these characters at the end of the builder.
- insert(): Adds the characters at a specified point.

Class Hierarchy	Syntax:

`java.lang.Object`
` ↳ java.lang`
`    ↳ Class StringBuilder`
	`public final class StringBuilder`
	`    extends Object`
	`    implements Serializable,`
	`CharSequence`

Exercise 1: Implement StringBuilder

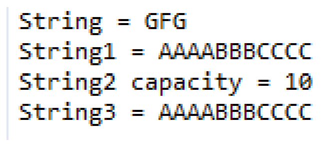
In this exercise you will understand the implementation of StringBuilder class.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyStringBuilderClass**.

```
// Java code to illustrate StringBuilder

public class MyStringBuilderClass {
        public static void main(String[] argv)
                throws Exception
        {

                // create a StringBuilder object
                // using StringBuilder() constructor
                StringBuilder str
                        = new StringBuilder();

                str.append("GFG");

                // print string
                System.out.println("String = "
                                                + str.toString());
                // create a StringBuilder object
                // usind StringBuilder(CharSequence) constructor
                StringBuilder str1
                        = new StringBuilder("AAAABBBCCCC");
                // print string
                System.out.println("String1 = "
                                               + str1.toString());

                // create a StringBuilder object
                // using StringBuilder(capacity) constructor
                StringBuilder str2
                        = new StringBuilder(10);

                // print string
                System.out.println("String2 capacity = "
                                               + str2.capacity());

                // create a StringBuilder object
                // usind StringBuilder(String) constructor
                StringBuilder str3
                        = new StringBuilder(str1);

                // print string
                System.out.println("String3 = "
                                               + str3.toString());
        }
}
```

5. Click **Run** to view the output.



```
String = GFG
String1 = AAAABBBCCCC
String2 capacity = 10
String3 = AAAABBBCCCC
```

StringBuffer

StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. The StringBuffer class provides a more efficient mechanism for building strings

The difference between String and StringBuffer are:

String	StringBuffer
String concatenation can get very expensive	
String concatenation is converted by most compilers	
If building a simple String, just concatenate	If building a String through a loop, use a StringBuffer

- String concatenation can get very expensive
- String concatenation is converted by most compilers — including Eclipse — into a StringBuffer implementation

- If building a simple String, just concatenate; if building a String through a loop, use a StringBuffer

  The StringBuffer class provides a more efficient mechanism for building strings

  String concatenation can get very expensive

  String concatenation is converted by most compilers — including Eclipse — into a StringBuffer implementation

  If building a simple String, just concatenate; if building a String through a loop, use a StringBuffer

Exercise 2: Implement StringBuffer

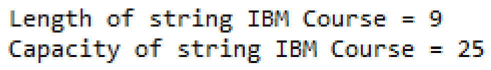
In this exercise you will understand the implementation of StringBuffer class.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyStringBufferClass**.

```
import java.io.*;
class MyStringBufferClass {
	public static void main(String[] args)
	{
		StringBuffer s = new StringBuffer("IBMCourse");
		int p = s.length();
		int q = s.capacity();
		System.out.println("Length of string IBM Course = " + p);
		System.out.println("Capacity of string IBM Course = " + q);
	}
}
```

5. Click **Run** to view the output.



```
Length of string IBM Course = 9
Capacity of string IBM Course = 25
```

Working with Date and Time

Using Date class

The class Date represents a specific instant in time, with millisecond precision. The Date class of java.util package implements Serializable, Cloneable and Comparable interface. It provides constructors and methods to deal with date and time with java.

Declaration:

```
public class Date
 extends Object
 implements Serializable, Cloneable, Comparable<Date>
```

Class Constructor:

- **Date()**
  This constructor allocates a Date object and initializes it so that it represents the time at which it was allocated, measured to the nearest millisecond.

- **Date(long Date)**
  This constructor allocates a Date object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.

Exercise 3: Implement Date

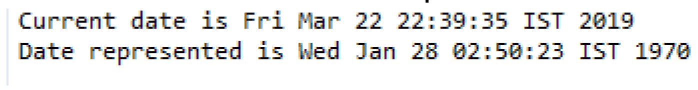
In this exercise you will understand the implementation of Date class.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyDateClass**.

```
// Java program to demonstrate constructors of Date
import java.util.*;

public class Main
{
	public static void main(String[] args)
	{
		Date d1 = new Date();
		System.out.println("Current date is " + d1);
		Date d2 = new Date(2323223232L);
		System.out.println("Date represented is "+ d2 );
	}
}
```

6. Click **Run** to view the output.



```
Current date is Fri Mar 22 22:39:35 IST 2019
Date represented is Wed Jan 28 02:50:23 IST 1970
```

SimpleDateFormat

SimpleDateFormat is a concrete class for formatting and parsing dates in a locale-sensitive manner. It allows for formatting (date -> text), parsing (text -> date), and normalization.

SimpleDateFormat allows you to start by choosing any user-defined patterns for date-time formatting. However, you are encouraged to create a date-time formatter with either getTimeInstance, getDateInstance, or getDateTimeInstance in DateFormat. Each of these class methods can return a date/time formatter initialized with a default format pattern. You may modify the format pattern using the applyPattern methods as desired.

Exercise 4: Implement Conversion

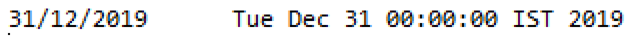
In this exercise you will understand the implementation of String to Date and Date to String conversion.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyStringToDateClass**.

```
// converting the String to Date format
import java.text.SimpleDateFormat;
import java.util.Date;
public class MyStringToDateClass {
   public static void main(String[] args)throws Exception {
      String sDate1="31/12/2019";
      Date date1=new
SimpleDateFormat("dd/MM/yyyy").parse(sDate1);
      System.out.println(sDate1+"\t"+date1);
   }
}
```

7. Click **Run** to view the output.



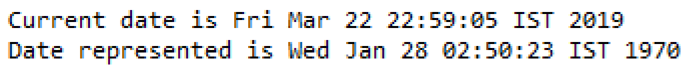
```
31/12/2019      Tue Dec 31 00:00:00 IST 2019
```

Procedure

5. From the project you have created in the eclipse IDE tool.
6. Click **File** > **Project** > **Java** > **Java Project**
7. Click **Next**.
8. Project name as **MyStringToDateClass**.

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Calendar;
class MyDateToStringClass{
    public static void main(String args[]){
          Date = Calendar.getInstance().getTime();
          DateFormat = new SimpleDateFormat("yyyy-mm-dd
hh:mm:ss");
          String strDate = dateFormat.format(date);
          System.out.println("Converted String: " + strDate);
    }
}
```

8. Click **Run** to view the output.



```
Current date is Fri Mar 22 22:59:05 IST 2019
Date represented is Wed Jan 28 02:50:23 IST 1970
```

Calendar Class

The Calendar class is an abstract class that provides methods for converting between a specific instant in time and a set of Calendar Fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR, and so on, and for manipulating the calendar fields, such as getting the date of the next week. An instant in time can be represented by a millisecond value that is an offset from the *Epoch*, January 1, 1970 00:00:00.000 GMT (Gregorian).

The class also provides additional fields and methods for implementing a concrete calendar system outside the package. Those fields and methods are defined as protected.

As it is an Abstract class, so we cannot use a constructor to create an instance. Instead, we will have to use the static method Calendar.getInstance() to instantiate and implement a sub-class.

- Calendar.getInstance(): return a Calendar instance based on the current time in the default time zone with the default locale.
- Calendar.getInstance(TimeZone zone)
- Calendar.getInstance(Locale aLocale)
- Calendar.getInstance(TimeZone zone, Locale aLocale)

Exercise 5: Implement Calendar

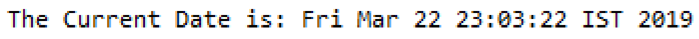
In this exercise you will understand the implementation of Calendar using getinstance method

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyCalendarClass**.

```
// converting the String to Date format
// Date getTime(): It is used to return a
// Date object representing this
// Calendar's time value.

import java.util.*;
public class MyCalendarClass {
        public static void main(String args[])
        {
                Calendar c = Calendar.getInstance();
                System.out.println("The Current Date is: " + c.getTime());
        }
}
```

9. Click **Run** to view the output.



```
The Current Date is: Fri Mar 22 23:03:22 IST 2019
```

Objects of primitives

Wrapper Classes

Wrapper classes are used for converting primitive data types into objects, like int to Integer etc.

- Primitives have no associated methods; there is no behavior associated with
    - primitive data types
    - Each primitive data type has a corresponding class, called a wrapper
    - Each wrapper object simply stores a single primitive variable and offers methods with which to process it
- Wrapper classes are included as part of the base Java API

Primitive Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Char
double	Double
float	Float
int	Int
long	Long
short	Short

Examples of using wrapper classes

double number = **Double.parseDouble("42.76");**
String hex = **Integer.toHexString(42);**
double value = **new Integer("1234").doubleValue();**
String input = "test 1-2-3";
int output = 0;
for (int index = 0; index < input.length(); index++) {
char c = input.charAt(index);
if (**Character.isDigit(c)**)

```
output = output * 10 + Character.digit(c, 10);
}
System.out.println(output) // 123
```

Wrapper classes, Autoboxing and unboxing

Autoboxing: Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

Exercise 6: Implement Autoboxing

In this exercise you will understand the implementation of autoboxing .

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyAutoboxingClass**.

```
// Java program to demonstrate Autoboxing

import java.util.ArrayList;
class MyAutoboxingClass
{
        public static void main(String[] args)
        {
                char ch = 'a';
                // Autoboxing- primitive to Character object conversion
                Character a = ch;

                ArrayList<Integer> arrayList = new ArrayList<Integer>();
                // Autoboxing because ArrayList stores only objects
                arrayList.add(25);

                // printing the values from object
                System.out.println(arrayList.get(0));
        }
}
```

5. Click **Run** to view the output.

```
25
```

Unboxing

It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.

Exercise 7: Implement UnAutoboxing

In this exercise you will understand the implementation of unautoboxing .

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyAutoboxingClass**.

```
// Java program to demonstrate Unboxing
import java.util.ArrayList;

class MyUnboxingClass
{
```

Java Training Module

```
        public static void main(String[] args)
        {
                Character ch = 'a';

                // unboxing - Character object to primitive conversion
                char a = ch;

                ArrayList<Integer> arrayList = new ArrayList<Integer>();
                arrayList.add(24);

                // unboxing because get method returns an Integer object
                int num = arrayList.get(0);

                // printing the values from primitive data types
                System.out.println(num);
        }
}
```

5. Click **Run** to view the output.

```
24
```

Test Your Knowledge

1. Which of these class is superclass of String and StringBuffer class?

   A. java.util
   B. java.lang
   C. ArrayList
   D. None of the mentioned

2. Which of this method of class StringBuffer is used to concatenate the string representation to the end of invoking string?

   A. concat()
   B. append()
   C. join()
   D. concatenate()

3. What is the output of this program?

```
class output {
        public static void main(String args[]) {
            StringBuffer c = new StringBuffer("Hello");
            c.delete(0,2);
            System.out.println(c);
        }
 }
```

   A. He
   B. Hel
   C. lo
   D. llo

4. Output of this program

Java Training Module

```
import java.util.*;
      class date {
         public static void main(String args[]) {
            Date obj = new Date();
            System.out.print(obj);
         }
 }
```

A. Prints Present Date
B. Runtime Error
C. Any Garbage Value
D. Prints Present Time & Date

5. What is the output of this program?

```
      import java.util.*;
      class Bitset {
         public static void main(String args[]) {
            BitSet obj1 = new BitSet(5);
            BitSet obj2 = new BitSet(10);
            for (int i = 0; i < 5; ++i)
               obj1.set(i);
            for (int i = 3; i < 13; ++i)
               obj2.set(i);
            obj1.and(obj2);
            System.out.print(obj1);
         }
 }
```

A. {0, 1}
B. {2, 4}
C. {3, 4}
D. {3, 4, 5}

Chapter 7 Exceptions

Introduction

Following are the features of the exceptions:

- An exception is an event or condition that disrupts the normal flow of execution in a program
- Exceptions are errors in a Java program
- The condition causes the system to throw an exception
- The flow of control is interrupted, and a handler will catch the exception
- Handling and declaring exceptions.

Checked and unchecked exception

Following are the two types of exception:

Checked Exceptions	A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions.
Unchecked Exceptions	An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions.

Java Approach to handle exceptions

A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following –

```
try {
  // Protected code
} catch (ExceptionName e1) {
  // Catch block
}
```

When an exception occurs, that exception occurred is handled by catch block associated with it. Every try block should be immediately followed either by a catch block or finally block.

Finally block

Finally, block is used when there are:

- Optional clause that allows cleanup and other operations to occur whether an exception occurs or not
- May have try/finally with no catch clauses
- Executed after any of the following:
    - try block completes normally
    - catch clause executes
    - Even if catch clause includes return.
- Unhandled exception is thrown, but before execution returns to calling method

Nested exception handling

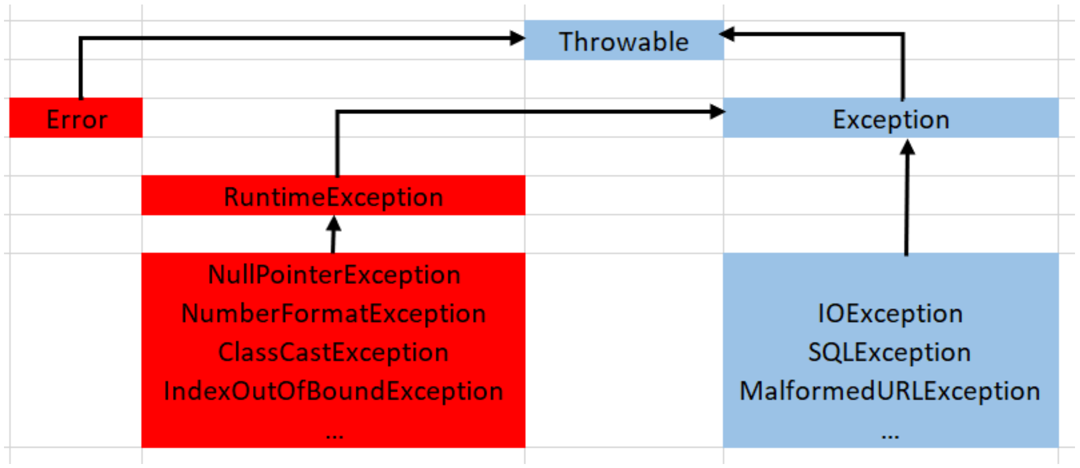
Following are the cases, where nested exception can be handled.

- It may be necessary to handle exceptions inside a catch or finally clause .
  - For example, you may want to log errors to a file, but all I/O operations require IOException to be caught
- Do this by nesting a try/catch (and optional finally) sequence inside your handler.

```
try{
//processing
 }catch(MyException){
  try {
         //Log Error
     } catch (IOExceptoin ioe) {
             ioe.printstacktrace();
      } finally {
         //close error log file
     }
}
```

Exception Hierarchy

Java exception classes are organized into a hierarchy. There is a basic exception class called Exception as you might expect. But in fact, the base of the hierarchy starts not with Exception but with a class called Throwable, which is then subclasses into Exception and Error. The diagram depicts the exception hierarchy.



The details of the hierarchy are:

- Exception subclasses represent errors that a program can reasonably recover from. Except for RuntimeException and its subclasses
- Error subclasses represent "serious" errors that a program generally shouldn't expect to catch and recover from.
- RuntimeException and its subclasses are slightly different: they represent exceptions that a program shouldn't generally expect to occur but could potentially recover from.

Exception Propagation

Propagation is a process in which the exception is being dropped from to the top to the bottom of the stack. If not caught once, the exception again drops down to the previous method and so on until it gets caught or until it reaches the very bottom of the call stack. This is called exception propagation, and this happens in case of Unchecked Exceptions.

Exercise 1: Implement Exception

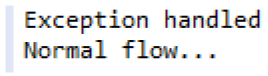
In this exercise you will understand the implementation of exception .

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MySimpleExceptionClass**.

```
// Java program to illustrate
// unchecked exception propagation
// without using throws keyword
class MySimpleExceptionClass {
        void m()
        {
                int data = 50 / 0; // unchecked exception occurred
                // exception propagated to n()
        }
        void n()
        {
                m();
                // exception propagated to p()
        }
        void p()
        {
                try {
                        n(); // exception handled
                }
                catch (Exception e) {
                        System.out.println("Exception handled");
                }
        }
        public static void main(String args[])
        {
                MySimpleExceptionClass obj = new
MySimpleExceptionClass();
                obj.p();
                System.out.println("Normal flow...");
        }
}
```

5. Click **Run** to view the output.



```
Exception handled
Normal flow...
```

The Throw Keyword

The features of throw keyword are:

- Can be used in a try block when you want to deliberately throw an exception
- You can throw a predefined Throwable object or your own Exception subtype
    - Create a new instance of the exception class to encapsulate the condition
- The flow of the execution stops immediately after the throw statement, and the next statement is not reached
- A finally clause is still executed if present.

Example

```
throw new java.io.IOException(“msg”);
```

Throwing Exceptions

An object to Throwable or to its sub classes can be explicitly created and thrown by using throw keyword.

Syntax

```
throw InstanceOfThrowableType;
```

where, InstanceOfThrowableType must be an object of type Throwable or subclass of Throwable.

Such explicitly thrown exception must be handled somewhere in the program, otherwise program will be terminated.

Rethrowing Exceptions

We all know that exceptions occurred in the try block are caught in catch block. Thus, caught exceptions can be re-thrown using throw keyword. Re-thrown exception must be handled somewhere in the program, otherwise program will terminate abruptly.

Try with resource Statement

The try-with-resources statement is a try statement that declares one or more resources. A resource is an object that must be closed after the program is finished with it. The try-with-resources statement ensures that each resource is closed at the end of the statement. Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource

```
static String readString path) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(path));
    try {
        return br.readLine();
    } finally {
        if (br != null) br.close();
    }
}
```

Exercise 2: Implement CheckedException

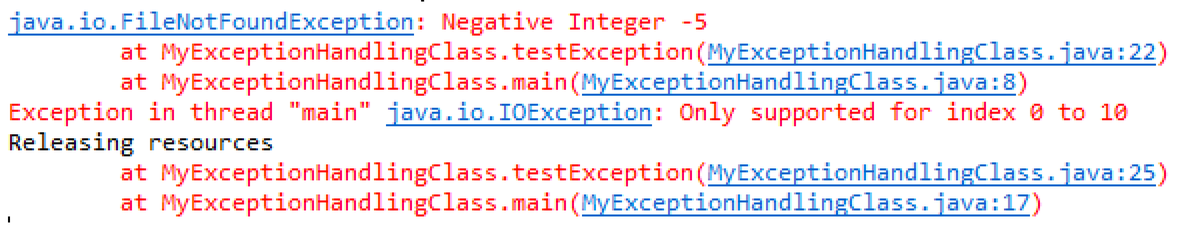
In this exercise you will understand the implementation of exception using throw keyword.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyExceptionHandlingClass**.

```
import java.io.FileNotFoundException;
import java.io.IOException;

public class MyExceptionHandlingClass {

public static void main(String[] args) throws FileNotFoundException,
IOException {
        try{
            testException(-5);
            testException(-10);
        }catch(FileNotFoundException e){
            e.printStackTrace();
        }catch(IOException e){
            e.printStackTrace();
        }finally{
            System.out.println("Releasing resources");
        }
        testException(15);
    }

    public static void testException(int i) throws FileNotFoundException,
IOException{
        if(i < 0){
            FileNotFoundException myException = new
FileNotFoundException("Negative Integer "+i);
            throw myException;
        }else if(i > 10){
            throw new IOException("Only supported for index 0 to 10");
        }
      }
   }
```

5. Click **Run** to view the output.



```
java.io.FileNotFoundException: Negative Integer -5
	at MyExceptionHandlingClass.testException(MyExceptionHandlingClass.java:22)
	at MyExceptionHandlingClass.main(MyExceptionHandlingClass.java:8)
Exception in thread "main" java.io.IOException: Only supported for index 0 to 10
Releasing resources
	at MyExceptionHandlingClass.testException(MyExceptionHandlingClass.java:25)
	at MyExceptionHandlingClass.main(MyExceptionHandlingClass.java:17)
```

Exercise 3: Implement UncheckedException

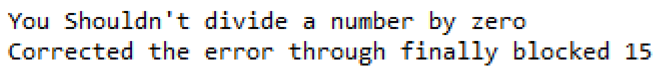
In this exercise you will understand the implementation of ArthimeticException class.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next.**
4. Project name as **MyUncheckedExceptionClass**.

```
public class MyUncheckedExceptionClass
{
  public static void main(String args[])
  {
          int num1 = 30, num2 = 0, output = 0;
    try{
      output=num1/num2;
      System.out.println ("Result: "+output);
    }
    catch(ArithmeticException e){
      System.out.println ("You Shouldn't divide a number by zero");
    }finally{
        num2=2;
      output=num1/num2;
      System.out.println("Corrected the error through finally blocked " +
output);
    }
  }
}
```

5. Click **Run** to view the output.



```
You Shouldn't divide a number by zero
Corrected the error through finally blocked 15
```

Exercise 4: Implement CustomException

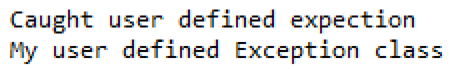
In this exercise you will understand the implementation of user defined exception.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next.**
4. Project name as **MyUserDefinedExceptionDemo**.

```
// A Class that represents use-defined exception
class MyUserDefinedException extends Exception
{
   public MyUserDefinedException(String s)
   {
      // Call constructor of parent Exception
      super(s);
   }
}
// A Class that uses above MyUserDefinedException
public class MyUserDefinedExceptionDemo
{
   // Driver Program
   public static void main(String args[])
   {
      try
      {
         // Throw an object of user defined exception
         throw new MyUserDefinedException("My user defined Exception
class");
      }
      catch (MyUserDefinedException ex)
      {
         System.out.println("Caught user defined expection");

         // Print the message from MyException object
         System.out.println(ex.getMessage());
      }
   }
}
```

5. Click **Run** to view the output.



```
Caught user defined expection
My user defined Exception class
```

Test Your Knowledge

1. Which of these exceptions handles the situations when an illegal argument is used to invoke a method?

   A. IllegalException
   B. Argument Exception
   C. IllegalArgumentException
   D. IllegalMethodArgumentException

2. Which of these class is used to create user defined exception?

   A. java.lang
   B. Exception
   C. RunTime
   D. System

3. What is the output of this program?

```
class exception_handling  {
        public static void main(String args[]) {
            try {
              int a[] = {1, 2,3 , 4, 5};
              for (int i = 0; i < 7; ++i)
                 System.out.print(a[i]);
            }catch(ArrayIndexOutOfBoundsException e) {
                System.out.print("0");
            }
        }
}
```

A. 12345
B. 123450
C. 1234500
D. Compilation Error

4. Which of these exceptions will be thrown if we use null reference for an arithmetic operation?

A. ArithmeticException
B. NullPointerException
C. IllegalAccessException
D. IllegalOperationException

5. Which of these classes is super class of Exception class?

A. Throwable
B. System
C. RunTime
D. Class

Chapter 8: File Handling

Getting information about files and folders

File Class in JAVA

An abstract representation of file and directory pathnames.

User interfaces and operating systems use system-dependent *pathname strings* to name files and directories. This class presents an abstract, system-independent view of hierarchical pathnames. An *abstract pathname* has two components:

- An optional system-dependent *prefix* string, such as a disk-drive specifier, "/" for the UNIX root directory, or "\\\\" for a Microsoft Windows UNC pathname, and
- A sequence of zero or more string *names*.
- A pathname, whether abstract or in string form can be either absolute or relative. The parent of an abstract pathname may be obtained by invoking the getParent() method of this class.
- First, we should create the File class object by passing the filename or directory name to it. A file system may implement restrictions to certain operations on the actual file-system object, such as reading, writing, and executing. These restrictions are collectively known as access permissions.
- Instances of the File class are immutable; that is, once created, the abstract pathname represented by a File object will never change.

A File object is created by passing in a String that represents the name of a file, or a String or another File object. For example,

File a = new File("<FILE_PATH>");

defines an abstract file name for the geeks file in directory <FILE_PATH>. This is an absolute abstract file name.

Exercise 1: Implement MyFolderContentsClass

In this exercise you will understand the implementation of user defined exception.

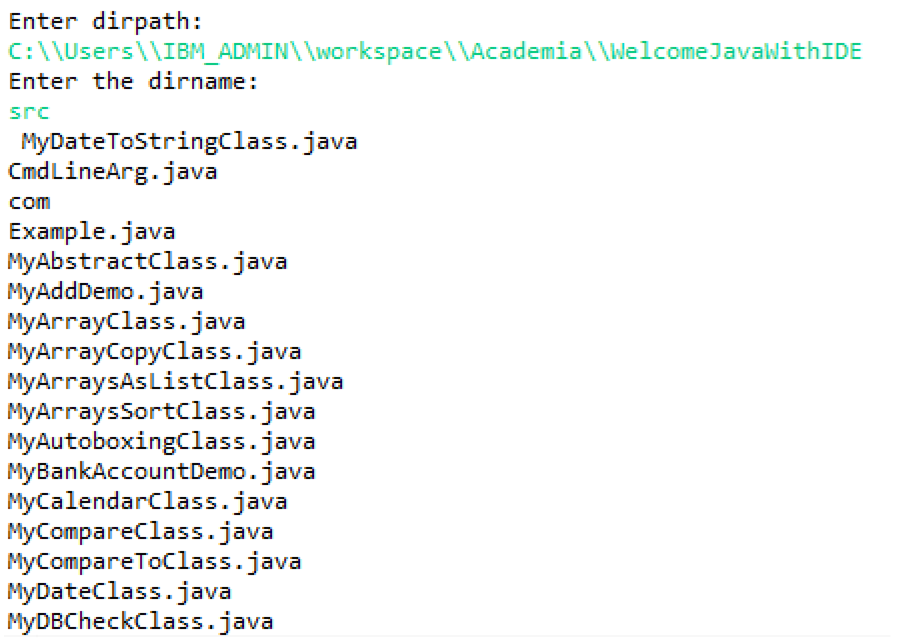
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyFolderContentsClass**.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;

//Displaying the contents of a directory
class MyFolderContentsClass
{
        public static void main(String[] args) throws IOException {
            //enter the path and dirname from keyboard
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

            System.out.println("Enter dirpath: ");
            String dirpath=br.readLine();
            System.out.println("Enter the dirname: ");
            String dname=br.readLine();

            //create File object with dirpath and dname
            File f = new File(dirpath, dname);
```

```
				//if directory exists, then
				if(f.exists())
				{
					//get the contents into arr[]
					//now arr[i] represent either a File or Directory
					String arr[]=f.list();

					//find no. of entries in the directory
					int n=arr.length;

					//displaying the entries
					for (int i = 0; i < n ; i++) {
						System.out.println(arr[i]);
						//create File object with the entry and test
						//if it is a file or directory
						File f1=new File(arr[i]);
						if(f1.isFile())
							System.out.println(": is a file");
						if(f1.isDirectory())
							System.out.println(": is a directory");
					}
					System.out.println("No of entries in this
directory "+n);
				}
				else
					System.out.println("Directory not found");
			}
}
```

5. Click **Run** to view the output. Enter the directory path, which you need to read, Enter the directory name to list the files. If the directory name is invalid, you will see error.



```
Enter dirpath:
C:\\Users\\IBM_ADMIN\\workspace\\Academia\\WelcomeJavaWithIDE
Enter the dirname:
src
 MyDateToStringClass.java
CmdLineArg.java
com
Example.java
MyAbstractClass.java
MyAddDemo.java
MyArrayClass.java
MyArrayCopyClass.java
MyArraysAsListClass.java
MyArraysSortClass.java
MyAutoboxingClass.java
MyBankAccountDemo.java
MyCalendarClass.java
MyCompareClass.java
MyCompareToClass.java
MyDateClass.java
MyDBCheckClass.java
```

File Class to Create Files and Directories

Create New File

You can create an empty file with an initial set of attributes by using the createFile (Path, FileAttributes<>)method. For example, if, at the time of creation, you want a file to have a set of file permissions, use the createFile method to do so. If you do not specify any attributes, the file is created with default attributes. If the file already exists, createFile throws an exception.

In a single atomic operation, the createFile method checks for the existence of the file and creates that file with the specified attributes, which makes the process more secure against malicious code.

Example

```
Path file = ...;
try {
    // Create the empty file with default permissions, etc.
    Files.createFile(file);
} catch (FileAlreadyExistsException x) {
    System.err.format("file named %s" +
        " already exists%n", file);
} catch (IOException x) {
    // Some other sort of failure, such as permissions.
    System.err.format("createFile error: %s%n", x);
}
```

Create New Directory

You can create a new directory by using the createDirectory(Path,FileAttributes<>) method. If you don't specify any FileAttributes, the new directory will have default attributes. For example:

Path dir = ...;
Files.createDirectory(path);

To create a directory several levels deep when one or more of the parent directories might not yet exist, you can use the convenience method, createDirectory(Path,FileAttributes<>). As with the createDirectory(Path, FileAttribute<?>) method, you can specify an optional set of initial file attributes. The following code snippet uses default attributes:

Files.createDirectories(Paths.get("foo/bar/test"));

The directories are created, as needed, from the top down. In the foo/bar/test example, if the foo directory does not exist, it is created. Next, the bar directory is created, if needed, and, finally, the test directory is created.

It is possible for this method to fail after creating some, but not all, of the parent directories.

Overview of streams API in JAVA

I/O Streams in JAVA

An *I/O Stream* represents an input source or an output destination. A stream can represent many kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.

Streams support many kinds of data, including simple bytes, primitive data types, localized characters, and objects. Some streams simply pass on data; others manipulate and transform the data in useful ways.

No matter how they work internally, all streams present the same simple model to programs that use them: A stream is a sequence of data. A program uses an *input stream* to read data from a source, one item at a time. A program uses an *output stream* to write data to a destination, one item at time

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream

Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams, but the most frequently used classes are, **FileInputStream** and **FileOutputStream**.

Following is an example which makes use of these two classes to copy an input file into an output file

Exercise 2: Implement FileInputStream

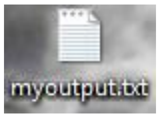
In this exercise you will understand the implementation of file input stream and file output stream class

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next.**
4. Project name as **MyCopyFileClass**.

```
import java.io.*;
public class MyCopyFileClass{

public static void main(String args[]) throws IOException {
    FileInputStream in = null;
    FileOutputStream out = null;

    try {
      in = new
FileInputStream("C:\\Users\\IBM_ADMIN\\Desktop\\myinput.txt");
      out = new
FileOutputStream("C:\\Users\\IBM_ADMIN\\Desktop\\myoutput.txt");

      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
      }
    }finally {
      if (in != null) {
        in.close();
      }
      if (out != null) {
        out.close();
      }
    }
  }
}
```

5. Click **Run** to view the output. Make sure the file is existing in the specified path to read the file. Also, cross check the file is created through file output stream class, as shown in the image.



Character Streams

Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams, but the most frequently used classes are, **FileReader** and **FileWriter**. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

We can re-write the above example, which makes the use of these two classes to copy an input file (having unicode characters) into an output file.

Buffered Streams

Most of the examples we've seen so far use *unbuffered* I/O. This means each read or write request is handled directly by the underlying OS. This can make a program much less efficient, since each such request often triggers disk access, network activity, or some other operation that is relatively expensive.

To reduce this kind of overhead, the Java platform implements *buffered* I/O streams. Buffered input streams read data from a memory area known as a *buffer*; the native input API is called only when the buffer is empty. Similarly, buffered output streams write data to a buffer, and the native output API is called only when the buffer is full.

A program can convert an unbuffered stream into a buffered stream using the wrapping idiom we've used several times now, where the unbuffered stream object is passed to the constructor for a buffered stream class. Here's how you might modify the constructor invocations in the CopyCharacters example to use buffered I/O:

```
inputStream = new BufferedReader(new FileReader("xanadu.txt"));
outputStream = new BufferedWriter(new
FileWriter("characteroutput.txt"));
```

There are four buffered stream classes used to wrap unbuffered streams: BufferedInputStream and BufferedOutputStream create buffered byte streams, while BufferedReader and BufferedWriter create buffered character streams.

Example of BuferredOutputStream

In this example, we are writing the textual information in the BufferedOutputStream object which is connected to the FileOutputStream object. The flush() flushes the data of one stream and send it into another. It is required if you have connected the one stream with another.

Exercise 3: Implement BufferedOutputStream

In this exercise you will understand the implementation of file input stream and file output stream class

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **BufferedOutputStreamClass**.

```
import java.io.*;
   public class BufferedOutputStreamClass{
   public static void main(String args[])throws Exception{
       FileOutputStream fout=new
FileOutputStream("<DIR_PATH>outfile.txt");
       BufferedOutputStream bout=new BufferedOutputStream(fout);
       String s="Welcome to ibm course";
       byte b[]=s.getBytes();
       bout.write(b);
       bout.flush();
       bout.close();
       fout.close();
       System.out.println("success");
   }
}
```

5. Click **Run** to view the output. Check the Outfile.txt file will contain “Welcome to ibm course”

```
success
```

Reading from Console

It has been becoming a preferred way for reading user's input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).
Advantages:

- Reading password without echoing the entered characters.
- Reading methods are synchronized.
- Format string syntax can be used.

Drawback:

- Does not work in non-interactive environment (such as in an IDE).

Scanner Class

The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however it is also can be used to read input from the user in the command line.

Advantages:

- Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
- Regular expressions can be used to find tokens.

Drawback:

- The reading methods are not synchronized.

Exercise 4: Implement Scanner

In this exercise you will understand the implementation of scanner class.

```
// Java program to demonstrate working of Scanner in Java
import java.util.Scanner;

class TakeInputFromUser
{
        public static void main(String args[])
        {
                // Using Scanner for Taking  Input from User
                Scanner in = new Scanner(System.in);

                String s = in.nextLine();
                System.out.println("You entered string "+s);

                int a = in.nextInt();
                System.out.println("You entered integer "+a);

                float b = in.nextFloat();
                System.out.println("You entered float "+b);
        }
}
```

Serialization

To *serialize* an object means to convert its state to a byte stream so that the byte stream can be reverted back into a copy of the object. A Java object is *serializable* if its class or any of its superclasses implements either the java.io.Serializable interface or its subinterface, java.io.Externalizable. *Deserialization* is the process of converting the serialized form of an object back into a copy of the object.

For example, the java.awt.Button class implements the Serializable interface, so you can serialize a java.awt.Button object and store that serialized state in a file. Later, you can read back the serialized state and deserialize into a java.awt.Button object.

Object serialization

Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.

After a serialized object has been written into a file, it can be read from the file and deserialized that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.

Most impressive is that the entire process is JVM independent, meaning an object can be serialized on one platform and deserialized on an entirely different platform.

Classes **ObjectInputStream** and **ObjectOutputStream** are high-level streams that contain the methods for serializing and deserializing an object.

The ObjectOutputStream class is used to serialize an Object. The following SerializeDemo program instantiates an Employee object and serializes it to a file.

Storing objects via serialization

When the program is done executing, a file named employee.ser is created. The program does not generate any output but study the code and try to determine what the program is doing.

Scenario

```
import java.io.*;
public class SerializeExample {

   public static void main(String [] args) {
      Employee e = new Employee();
      e.name = "Raj Malhotra";
      e.address = "IBM India, Bangalore";
      e.SSN = 1234256;
      e.number = 071423Q;

      try {
         FileOutputStream fileOut =
         new FileOutputStream("<DIR_PATH>\employee.ser");
         ObjectOutputStream out = new ObjectOutputStream(fileOut);
         out.writeObject(e);
         out.close();
         fileOut.close();
         System.out.printf("Serialized data is saved in <DIR_PATH>/employee.ser");
      } catch (IOException i) {
         i.printStackTrace();
      }
   }
}
```

Retrieving serialized Object

The following DeserializeExample program deserializes the Employee object created in the SerializeExample program.

Scenario

```
import java.io.*;
public class DeserializeExample {

   public static void main(String [] args) {
      Employee e = null;
      try {
         FileInputStream fileIn = new FileInputStream("<DIR_PATH>/employee.ser");
         ObjectInputStream in = new ObjectInputStream(fileIn);
         e = (Employee) in.readObject();
         in.close();
         fileIn.close();
      } catch (IOException i) {
         i.printStackTrace();
         return;
      } catch (ClassNotFoundException c) {
         System.out.println("Employee class not found");
         c.printStackTrace();
         return;
      }

      System.out.println("Deserialized Employee...");
      System.out.println("Name: " + e.name);
      System.out.println("Address: " + e.address);
      System.out.println("SSN: " + e.SSN);
      System.out.println("Number: " + e.number);
   }
}
```

Metadata and File attributes

The definition of *metadata* is "data about other data." With a file system, the data is contained in its files and directories, and the metadata tracks information about each of these objects: Is it a regular file, a directory, or a link? What is its size, creation date, last modified date, file owner, group owner, and access permissions?

A file system's metadata is typically referred to as its *file attributes*. The Files class includes methods that can be used to obtain a single attribute of a file, or to set an attribute.

Methods	**Comment**
size(Path)	Returns the size of the specified file in bytes.
isDirectory(Path, LinkOption)	Returns true if the specified `Path` locates a file that is a directory.
isRegularFile(Path, LinkOption...)	Returns true if the specified `Path` locates a file that is a regular file.
isSymbolicLink(Path)	Returns true if the specified `Path` locates a file that is a symbolic link.
isHidden(Path)	Returns true if the specified `Path` locates a file that is considered hidden by the file system.

getLastModifiedTime(Path, LinkOption...)
setLastModifiedTime(Path, FileTime)
	Returns or sets the specified file's last modified time.

getOwner(Path, LinkOption...)
setOwner(Path, UserPrincipal)
	Returns or sets the owner of the file.

getPosixFilePermissions(Path, LinkOption...)	Returns or sets a file's POSIX file permissions.

setPosixFilePermissions(Path, Set<PosixFilePermission>)	
getAttribute(Path, String, LinkOption...) setAttribute(Path, String, Object, LinkOption...)	Returns or sets the value of a file attribute.

If a program needs multiple file attributes around the same time, it can be inefficient to use methods that retrieve a single attribute. Repeatedly accessing the file system to retrieve a single attribute can adversely affect performance. For this reason, the `Files` class provides two `readAttributes` methods to fetch a file's attributes in one bulk operation.

Method	Comment
`readAttributes(Path, String, LinkOption...)`	Reads a file's attributes as a bulk operation. The `String` parameter identifies the attributes to be read.
`readAttributes(Path, Class<A>, LinkOption...)`	Reads a file's attributes as a bulk operation. The `Class<A>` parameter is the type of attributes requested and the method returns an object of that class.

Basic File Attribute

to read the basic attributes of a file, you can use one of the `Files.readAttributes` methods, which reads all the basic attributes in one bulk operation. This is far more efficient than accessing the file system separately to read each individual attribute. The varargs argument currently supports the `LinkOption` enum, `NOFOLLOW_LINKS`. Use this option when you do not want symbolic links to be followed.

The following code snippet reads and prints the basic file attributes for a given file and uses the methods in the `BasicFileAttributes` class.

Symbolic and Hard links

- A symbolic link contains a reference to another file or directory.
- The file referenced by a symbolic link is known as the target file for the symbolic link.
- Operations on a symbolic link are transparent to the application. We can work with symbolic links using the java.nio.file.Files class.
- isSymbolicLink(Path p) method checks if the file specified by the specified path is a symbolic link.
- createSymbolicLink() method of the Files, which may not be supported on all platforms, creates a symbolic link.
- The Java NIO API follows the symbolic link by default. We can specify whether we want to follow a symbolic link or not. The option not to follow a symbolic link is indicated by using the enum constant LinkOption. NOFOLLOW_LINKS.
- The LinkOption enum is declared in the java.nio.file package. Methods supporting this option let we pass an argument of the LinkOption type.
- We can use the createLink(Path newLink, Path existingPath) method of the Files class to create a hard link.

Scenario

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
public class Main {
  public static void main(String[] args) throws Exception {
    Path existingFilePath = Paths.get("<DIR_PATH>/test1.txt");
    Path symLinkPath = Paths.get("/opt/ibm/test1_link.txt");
    Files.createSymbolicLink(symLinkPath, existingFilePath);
  }}
```

Use the method `createLink` to create a hard link. Following example shows how easy it is to create a hard link using `Files` class. We used `exists` method to check whether any link was created.

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
public class Main {
  public static void main(String[] args) throws Exception {

Path target = Paths.get("<DIR_PATH>/testhard.txt");
Path link = Paths.get("<DIR_PATH>/links/testhard_link.txt");

// creates hard link
Files.createLink(link, target);

// returns true

Files.exists(link);

  }
}
```

File Visitor Interface

To walk a file tree, you first need to implement a FileVisitor. A FileVisitor specifies the required behavior at key points in the traversal process: when a file is visited, before a directory is accessed, after a directory is accessed, or when a failure occurs. The interface has four methods that correspond to these situations:

- preVisitDirectory – Invoked before a directory's entries are visited.
- postVisitDirectory – Invoked after all the entries in a directory are visited. If any errors are encountered, the specific exception is passed to the method.
- visitFile – Invoked on the file being visited. The file's BasicFileAttributes is passed to the method, or you can use the file attributes package to read a specific set of attributes. For example, you can choose to read the file's DosFileAttributeView to determine if the file has the "hidden" bit set.
- visitFileFailed – Invoked when the file cannot be accessed. The specific exception is passed to the method. You can choose whether to throw the exception, print it to the console or a log file, and so on.

If you don't need to implement all four of the FileVisitor methods, instead of implementing the FileVisitor interface, you can extend the SimpleFileVisitor class. This class, which implements the FileVisitor interface, visits all files in a tree and throws an IOError when an error is encountered. You can extend this class and override only the methods that you require.

Random Access File

The **Java.io.RandomAccessFile** class file behaves like a large array of bytes stored in the file system. Instances of this class support both reading and writing to a random-access file.

There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of bytes has been read than EOFException is thrown. It is a type of IOException. Text for this section should be provided here.

Constructor

Constructor	Description
RandomAccessFile(File file, String mode)	Creates a random-access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile(String name, String mode)	Creates a random-access file stream to read from, and optionally to write to, a file with the specified name.

Scenario

```
_   import java.io.IOException;
    import java.io.RandomAccessFile;

    public class RandomAccessFileExample {
        static final String FILEPATH ="myFile.TXT";
        public static void main(String[] args) {
            try {
                System.out.println(new String(readFromFile(FILEPATH, 0, 18)));
                writeToFile(FILEPATH, "I am Studying JAVA", 31);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

Java Training Module

```
        private static byte[] readFromFile(String filePath, int position, int
size)
                throws IOException {
            RandomAccessFile file = new RandomAccessFile(filePath, "r");
            file.seek(position);
            byte[] bytes = new byte[size];
            file.read(bytes);
            file.close();
            return bytes;
        }
        private static void writeToFile(String filePath, String data, int
position)
                throws IOException {
            RandomAccessFile file = new RandomAccessFile(filePath, "rw");
            file.seek(position);
            file.write(data.getBytes());
            file.close();
        }
    }
```

Test Your Knowledge

1. Which of these class contains the methods used to write in a file?

   A. FileStream
   B. FileInputStream
   C. BUfferedOutputStream
   D. FileBufferStream

2. Which of these exceptions is thrown in cases when the file specified for writing is not found?

   A. IOException
   B. FileException
   C. FileNotFoundException
   D. FileInputException

3. Which of these exceptions is thrown by close() and read() methods?

   A. IOException
   B. FileException
   C. FileNotFoundException
   D. FileInputOutputException

4. What is the output of this program?

```
import java.io.*;
      class filesinputoutput {
         public static void main(String args[]) {
            InputStream obj = new FileInputStream("inputoutput.java");
            System.out.print(obj.available());
         }
  }
```

Note: inputoutput.java is stored in the disk.

A. true
B. false
C. prints number of bytes in file
D. prints number of characters in the file

5. Which of these methods are used to read in from file?

A. get()
B. read()
C. scan()
D. readFileInput()

Chapter 9: MultiThreading

Overview of Multithreading

Process and java threads

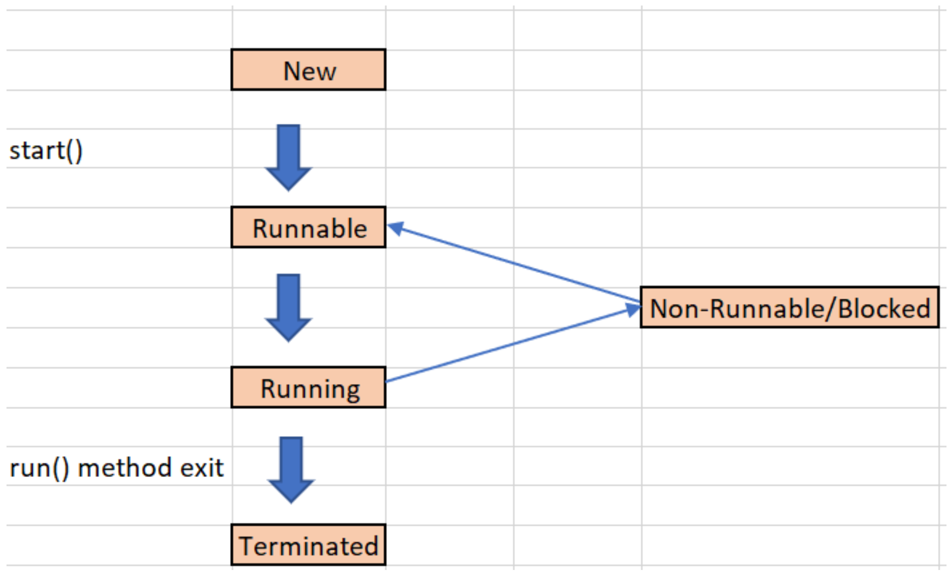
A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. So, threads are light-weight processes within a process.

Thread Lifecycle

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

New	The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
Runnable	After a newly born thread is started, the thread becomes runnable. A thread in this state is executing its task.
Running	The thread is in running state if the thread scheduler has picked it.
Blocked/Non-Runnable	A java thread can be blocked when waiting for a resource.
Terminated	A thread is in terminated or dead state when its run() method exits.

The flow diagram represents thread lifecycle.



Creating Threads

The two types of creating threads are:

1. Extending the thread class
2. Implementing the runnable interface

1. Extending the thread class

Thread can be created by extending class to Thread class. This class overrides the run() method available in the Thread class.

Exercise 1: Implement Thread

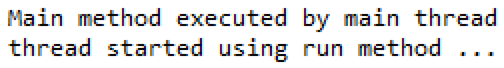
In this exercise you will understand the implementation of thread using extend keyword.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next.**
4. Project name as **MyThreadClass.**

```
public class MyThreadClass extends Thread{
        public void run() {
                System.out.println("thread started using run method ...");
        }

        public static void main(String[] args) {
                MyThreadClass t1 = new MyThreadClass();
                t1.start();
                 System.out.println("Main method executed by main
thread");
        }
}
```

5. Click **Run** to view the output.



```
Main method executed by main thread
thread started using run method ...
```

2. Implement the runnable interface

A new class which implements java.lang.Runnable interface and override run() method is created. Then we instantiate a Thread object and call start() method on this object.

Exercise 2: Implement Thread through runnable

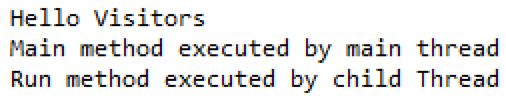
In this exercise you will understand the implementation of thread using runnable interface.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyThreadRunnableClass**.

```
class MyThreadRunnableTest {
   public static void m1()
   {
      System.out.println("Hello Visitors");
   }
}

// Here we can extends any other class
public class MyThreadRunnableClass extends MyThreadRunnableTest
implements Runnable {
   public void run()
   {
      System.out.println("Run method executed by child Thread");
   }
   public static void main(String[] args)
   {
        MyThreadRunnableClass t = new MyThreadRunnableClass();
      t.m1();
      Thread t1 = new Thread(t);
      t1.start();
      System.out.println("Main method executed by main thread");
   }
}
```

5. Click **Run** to view the output.



```
Hello Visitors
Main method executed by main thread
Run method executed by child Thread
```

Synchronization

- The synchronized keyword may be used to synchronize access to an object among the threads using the object
    - The synchronized keyword guards' critical sections of code
        - Either methods or arbitrary sections of code may be synchronized.
- All synchronized sections of code in an object are locked when a thread executes any one synchronized code section
    - No other threads can enter a synchronized section of code while it is locked.
    - Threads may still access other non-synchronized methods
- If the synchronized method is static, a lock is obtained at the class level instead of the object level.
    - Only one thread at a time may use such a method.

Exercise 3: Implement synchronize

In this exercise you will understand the implementation of thread using synchronize keyword.

Procedure

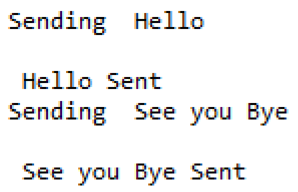
1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MySynchronizedClass**.

```
// A Java program to demonstrate working of synchronized.
import java.io.*;
import java.util.*;

// A Class used to send a message
class MessageSender
{
   public void send(String msg)
   {
      System.out.println("Sending\t"  + msg);
      try
      {
         Thread.sleep(1000);
      }
      catch (Exception e)
      {
         System.out.println("Thread  interrupted.");
      }
      System.out.println("\n" + msg + "Sent");
   }
}
```

Java Training Module

```
// Class for sending a message using Threads
class ThreadedSend extends Thread
{
   private String msg;
   private Thread t;
   MessageSender  msender;

   // Receives a message object and a string message to be sent
   ThreadedSend(String m,  MessageSender obj)
   {
      msg = m;
      msender = obj;
   }

   public void run()
   {
      // Only one thread can send a message
      // at a time.
      synchronized(msender)
      {
         // synchronizing the send object
         msender.send(msg);
      }
   }
}
// Driver class
class MySynchronizedClass
{
   public static void main(String args[])
   {
        MessageSender snd = new MessageSender();
      ThreadedSend S1 =
         new ThreadedSend( " Hello " , snd);
      ThreadedSend S2 =
         new ThreadedSend( " See you Bye " , snd);

      // Start two threads of ThreadedSend type
      S1.start();
      S2.start();

      // wait for threads to end
      try
      {
         S1.join();
         S2.join();
      }
      catch(Exception e)
      {
         System.out.println("Interrupted");
      }
   }
}
```

5. Click **Run** to view the output.



```
Sending  Hello

 Hello Sent
Sending  See you Bye

 See you Bye Sent
```

Synchronization Issues

- Use synchronization sparingly
    - Can slow performance by reducing concurrency.
    - Can sometimes lead to fatal conditions such as deadlock.
- Other techniques should be used with synchronization to assure optimal performance and to assist threads in coordinating their activities.
    - For example,
        - notifyAll() and
        - wait()

Following are the need for thread synchronization:

- In many situations, concurrently running threads must share data and consider the state and activities of other threads
    - Example: producer-consumer programming scenarios
- Producer thread generates data that is needed and consumed by another thread.
    - Data may be shared using a common object that both threads access
- In Java, an object can be operated on by multiple threads; it is the responsibility of the object to protect itself from any possible interference.
- Objects can be locked to prevent critical sections of code from being simultaneously accessed by multiple threads.

Race Condition

A race condition is a situation in which two or more threads or processes are reading or writing some shared data, and the result depends on the timing of how the threads are scheduled.

Running more than one thread inside the same application does not by itself cause problems. The problems arise when multiple threads access the same resources. It is safe to let multiple threads read the same resources, if the resources do not change.

Preventing Race condition

- Race conditions can be avoided by proper thread synchronization in critical sections.
- Thread synchronization can be achieved using locks or synchronized blocks of code.

Interthread communication

Interthread communication happens when two or more threads share information. Following keywords are used for interthread communication.

- Wait(),notify() and notifyAll() methods of object class are used for this purpose.
- Wait()- tells calling thread to give up monitor and go to sleep until some other thread enters the same monitor and call notify.
- Notify() - wakes up a thread that called wait() on same object.
- notifyAll() - wakes up all the thread that called wait() on same object.

Exercise 4: Implement inter-thread communication

In this exercise you will understand the implementation of inter-thread communication using wait, notify keywords, thread functions with runnable interface.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyWaitNotifyClass**.

```
import java.util.Scanner;
public class MyWaitNotifyClass
{
   public static void main(String[] args)
                        throws InterruptedException
   {
      final PC = new PC();

      // Create a thread object that calls pc.produce()
      Thread t1 = new Thread(new Runnable()
      {
         //Override run method
         public void run()
         {
            try
            {
               pc.produce();
            }
            catch(InterruptedException e)
            {
               e.printStackTrace();
```

Java Training Module

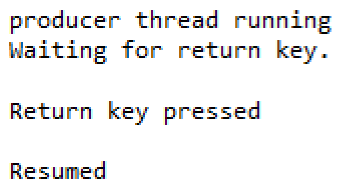
```
            }
        }
    });

    // Create another thread object that calls
    // pc.consume()
    Thread t2 = new Thread(new Runnable()
    {
        //Override run method
        public void run()
        {
            try
            {
                pc.consume();
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    });

    // Start both threads
    t1.start();
    t2.start();

    // t1 finishes before t2
    t1.join();
    t2.join();
}

// PC (Produce Consumer) class with produce() and
// consume() methods.
```

Java Training Module

```
public static class PC
{
   // Prints a string and waits for consume()
   public void produce()throws InterruptedException
   {
      // synchronized block ensures only one thread
      // running at a time.
      synchronized(this)
      {
         System.out.println("producer thread running");

         // releases the lock on shared resource
         wait();

         // and waits till some other method invokes notify().
         System.out.println("Resumed");
      }
   }

   // Sleeps for some time and waits for a key press. After key
   // is pressed, it notifies produce().
   public void consume()throws InterruptedException
   {
      // this makes the produce thread to run first.
      Thread.sleep(1000);
      Scanner s = new Scanner(System.in);

      // synchronized block ensures only one thread
      // running at a time.
      synchronized(this)
      {
         System.out.println("Waiting for return key.");
         s.nextLine();
```

```
                System.out.println("Return key pressed");

                // notifies the produce thread that it
                // can wake up.
                notify();

                // Sleep
                Thread.sleep(2000);
            }
        }
    }
}
```

5. Click **Run** to view the output.



```
producer thread running
Waiting for return key.

Return key pressed

Resumed
```

Test Your Knowledge

1. Which of these are types of multitasking?

   A. Process based
   B. Thread based
   C. Process and Thread based
   D. None of the mentioned

2. What will happen if two thread of the same priority are called to be processed simultaneously?

   A. Anyone will be executed first lexographically
   B. Both will be executed simultaneously
   C. None of them will be executed
   D. It is dependent on the operating system

3. What is the name of the thread in output of this program?

```
class multithreaded_programing {
        public static void main(String args[]){
            Thread t = Thread.currentThread();
            System.out.println(t);
         }
 }
```

   A. main
   B. Thread
   C. System
   D. None of the mentioned

4. Which of this method is used to find out that a thread is still running or not?

A. run()
B. Alive()
C. isAlive()
D. checkRun()

5. Which of these methods is used to explicitly set the priority of a thread?

A. set()
B. make()
C. setPriority()
D. makePriority()

Chapter 10: Collection Framework

Collections are objects that group multiple elements and store, retrieve, and manipulate those elements. The Collection interface is at the root of the collection hierarchy. Sub interfaces of Collection include List, Queue, and Set

Generics

What are Generics?

Generics allow type to be a parameter to methods, classes and interfaces.

Syntax

```
BaseType <Type> obj = new BaseType <Type>()
```

In Parameter type we cannot use primitives like 'int', 'char' or 'double'.

Exercise 1: Implement Generic

In this exercise you will understand the implementation of generic .

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyGenericsClass**.

```
import java.util.*;
class MyGenericsClass{
public static void main(String args[]){
        ArrayList<String> list=new ArrayList<String>();
        list.add("Sam");
        list.add("Derby");
        //list.add(32);//compile time error
                String s=list.get(1);//type casting is not required
        System.out.println("element is: "+s);
        Iterator<String> itr=list.iterator();
                while(itr.hasNext()){
                        System.out.println(itr.next());
                }
        }
}
```

5. Click **Run** to view the output.

```
element is: Derby
Sam
Derby
```

Bounded types

There may be times when you'll want to restrict the kinds of types that can be passed to a type parameter. For example, a method that operates on numbers might only want to accept instances of Number or its subclasses. This is what bounded type parameters are for.

```
class Box<T extends Number> { ... }
```

Overview of collection framework

Need for collection framework in java

- Java Collections Framework provides lots of different useful data types, such as linked lists (allows insertion anywhere in constant time), resizable array lists (like Vector but cooler), red-black trees, hash-based maps (like Hashtable but cooler).
- Java Collections Framework provides abstractions, so you can refer to a list as a List, whether backed by an array list or a linked list; and you can refer to a map/dictionary as a Map, whether backed by a red-black tree or a hashtable.

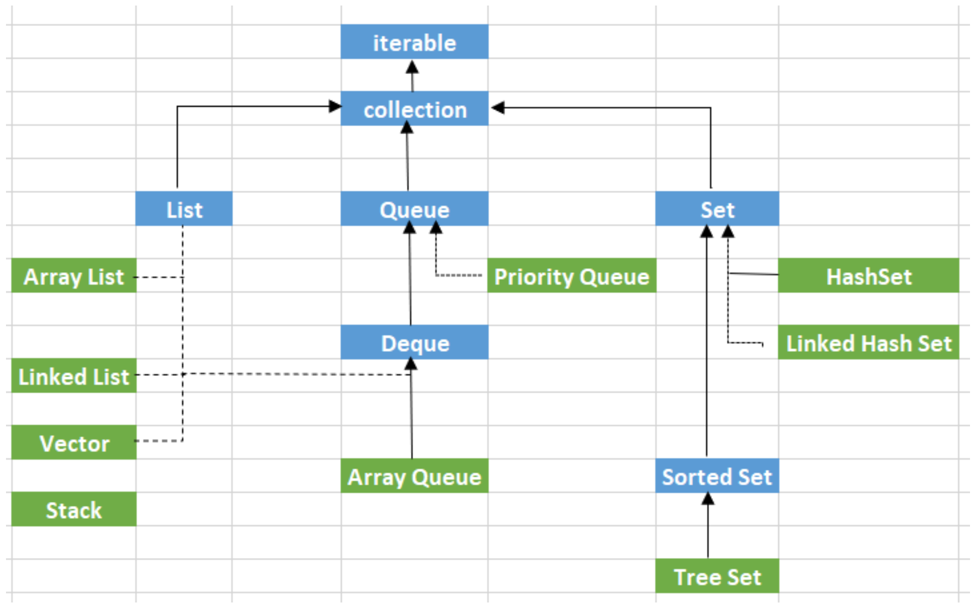
In other words, Java Collections Framework allows you to use the right data structure, because one size does not fit all.

The Collection hierarchy

The Collection framework represents a unified architecture for storing and manipulating a group of objects . it has

1.Interfaces and its implementations, i.e. , classes

2. Algorithm



The List interface and common implementations

List is an ordered collection. Lists may contain duplicate elements. Elements can be inserted or accessed by their position in the list, using a zero-based index.

Arraylist

Arraylist class implements List interface and it is based on an Array data structure.

Syntax

```
ArrayList<String> alist=new ArrayList<String>();
```

Add elements to Arraylist

We can use add() method to add elements to arraylist.

```
alist.add(“added”);
```

Remove from arraylist

Use remove() method to remove elements from arraylist.

```
alist.remove(“removed”);
```

Vector and stack

Vector

Vector implements List Interface. Like ArrayList it also maintains insertion order, but it is rarely used in non-thread environment as it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements.

Declaration

```
public class Vector<E>
extends Object<E>
implements List<E>, Cloneable, Serializable
```

Exercise 2: Implement Vector

In this exercise you will understand the implementation of vector.

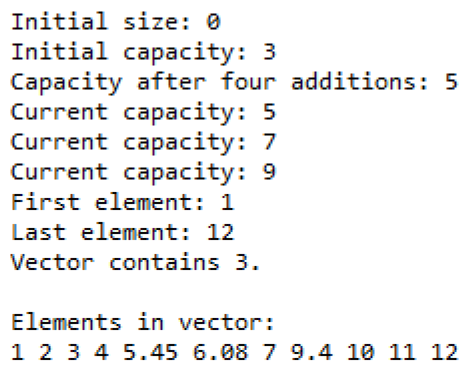
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyVectorClass**.

```
import java.util.*;

public class MyVectorClass {
          public static void main(String args[]) {
              // initial size is 3, increment is 2
              Vector v = new Vector(3, 2);
              System.out.println("Initial size: " + v.size());
              System.out.println("Initial capacity: " + v.capacity());

              v.addElement(new Integer(1));
              v.addElement(new Integer(2));
              v.addElement(new Integer(3));
              v.addElement(new Integer(4));
              System.out.println("Capacity after four additions: " +
v.capacity());

              v.addElement(new Double(5.45));
              System.out.println("Current capacity: " + v.capacity());

              v.addElement(new Double(6.08));
              v.addElement(new Integer(7));
              System.out.println("Current capacity: " + v.capacity());
```

```
            v.addElement(new Float(9.4));
            v.addElement(new Integer(10));
            System.out.println("Current capacity: " + v.capacity());

            v.addElement(new Integer(11));
            v.addElement(new Integer(12));
            System.out.println("First element: " +
(Integer)v.firstElement());
            System.out.println("Last element: " +
(Integer)v.lastElement());

            if(v.contains(new Integer(3)))
              System.out.println("Vector contains 3.");

            // enumerate the elements in the vector.
            Enumeration vEnum = v.elements();
            System.out.println("\nElements in vector:");

            while(vEnum.hasMoreElements())
              System.out.print(vEnum.nextElement() + " ");
            System.out.println();
          }
}
```

5. Click **Run** to view the output.



```
Initial size: 0
Initial capacity: 3
Capacity after four additions: 5
Current capacity: 5
Current capacity: 7
Current capacity: 9
First element: 1
Last element: 12
Vector contains 3.

Elements in vector:
1 2 3 4 5.45 6.08 7 9.4 10 11 12
```

Stack

Java provides a Stack class which models the Stack data structure. A Stack is a Last In First Out (LIFO) data structure. It supports two basic operations called **push** and **pop**.

The Stack class extends Vector which implements the List interface. A Vector is a re-sizable collection. It grows its size to accommodate new elements and shrinks the size when the elements are removed.

Since the Stack class extends Vector, it also grows and shrinks its size as needed when new elements are added or removed.

Exercise 3: Implement Stack

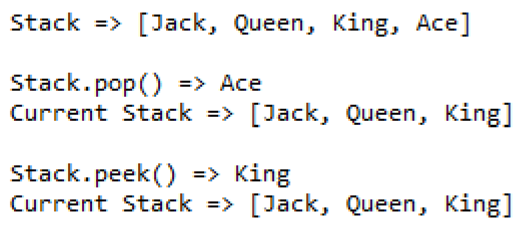
In this exercise you will understand the implementation of stack.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyStackClass**.

```
import java.util.Stack;

public class MyStackClass {
        public static void main(String args[]) {
                // Creating a Stack
                Stack<String> stackOfCards = new Stack<>();

                // Pushing new items to the Stack
                stackOfCards.push("Jack");
                stackOfCards.push("Queen");
                stackOfCards.push("King");
                stackOfCards.push("Ace");
                System.out.println("Stack => " + stackOfCards);
                System.out.println();

                // Popping items from the Stack
                String cardAtTop = stackOfCards.pop(); // Throws
EmptyStackException if the stack is empty
                System.out.println("Stack.pop() => " + cardAtTop);
                System.out.println("Current Stack => " + stackOfCards);
                System.out.println();

                // Get the item at the top of the stack without removing it
                cardAtTop = stackOfCards.peek();
                System.out.println("Stack.peek() => " + cardAtTop);
                System.out.println("Current Stack => " + stackOfCards);
        }
}
```

5. Click **Run** to view the output.



```
Stack => [Jack, Queen, King, Ace]

Stack.pop() => Ace
Current Stack => [Jack, Queen, King]

Stack.peek() => King
Current Stack => [Jack, Queen, King]
```

Iterators

Iterable and Iterator

Iterators are used in Collection framework in Java to retrieve elements one by one.

- To implement an iterable data structure, we need to:
- Implement Iterable interface along with its methods in the said Data Structure
- Create an Iterator class which implements Iterator interface and corresponding methods.
- The “Iterable” was introduced to be able to use in the “foreach” loop. A class implementing the Iterable interface can be iterated over.

Syntax

```
public interface Collection<E> extends Iterable<E>
```

Exercise 4: Implement IterableVsIterator

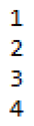
In this exercise you will understand the implementation of iterable and iterator using wrapper class integer.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyIterableVsIteratorClass**.

```
import java.util.Arrays;
import java.util.List;

public class MyIterableVsIteratorClass {
   public static void main(String[] args) {
      //numbers is iterable as List<E> extends Collection<E>
      //Collection<E> extends Iterable<E>
      List<Integer> numbers = Arrays.asList(new Integer[]{1,2,3,4});
      for (Integer integer : numbers) {
        System.out.println(integer);
      }
   }
}
```

5. Click **Run** to view the output.



```
1
2
3
4
```

Iterators

Iterator is class that manages iteration over an Iterable. It maintains a state of where we are in the current iteration and knows what the next element is and how to get it.

In Java 8, when you call the double “**::**” notation as shown below, you get an Iterable from the Stream.

Exercise 4: Implement Reader

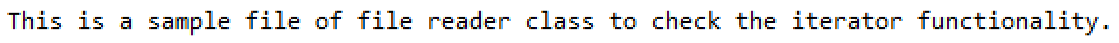
In this exercise you will understand the implementation of reader class.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyReadFileClass**.

```
import java.io.FileReader;
import java.io.IOException;

public class MyReadFileClass{
        public static void main(String[] args) throws IOException {
                // pass the path to the file as a parameter
                FileReader fr = new
FileReader("C:\\Users\\IBM_ADMIN\\Desktop\\myreadme.txt");
                int i;
                while ((i = fr.read()) != -1) {
                        System.out.print((char) i);
                }
        }
}
```

5. Click **Run** to view the output.



```
This is a sample file of file reader class to check the iterator functionality.
```

Using iterators in Java

Iterator is an interface which belongs to collection framework. It allows us to traverse the collection, access the data element and remove the data elements of the collection.
java.util package has public interface Iterator and contains three methods:

1. boolean hasNext(): It returns true if Iterator has more element to iterate.
2. Object next(): It returns the next element in the collection until the hasNext()method return true. This method throws NoSuchElementException if there is no next element.
3. void remove(): It removes the current element in the collection. This method throws IllegalStateException if this function is called before next( ) is invoked.

Exercise 5: Implement IteratorList

In this exercise you will understand the implementation of Iterator with the list interface.

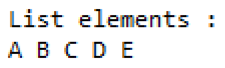
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyIteratorListClass**.

```
// Java code to illustrate the use of iterator
import java.io.*;
import java.util.*;

public class MyIteratorListClass{
   public static void main(String[] args){
      ArrayList<String> list = new ArrayList<String>();
```

```
    list.add("A");
    list.add("B");
    list.add("C");
    list.add("D");
    list.add("E");

    // Iterator to traverse the list
    Iterator iterator = list.iterator();

    System.out.println("List elements : ");

    while (iterator.hasNext())
      System.out.print(iterator.next() + " ");

    System.out.println();
  }
}
```

5. Click **Run** to view the output.



```
List elements :
A B C D E
```

Utility Class

Collections utility class

The java.util.Collections class consists exclusively of static methods that operate on or return collections. Following are the important points about Collections –

- It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection.
- The methods of this class all throw a NullPointerException if the collections or class objects provided to them are null.

Some useful method in Collections class:

Method Signature	Description
Collections.sort(List myList)	Sort the myList (implementation of any List interface) provided an argument in natural ordering.
Collections.sort(List, comparator c)	Sort the myList(implementation of any List interface) as per comparator c ordering. (c class should implement comparator interface)
Collections.shuffle(List myList)	Puts the elements of myList ((implementation of any List interface) in random order.
Collections.reverse(List myList)	Reverses the elements of myList ((implementation of any List interface)
Collections.binarySearch(List mlist, T key)	Searches the mlist (implementation of any List interface) for the specified object using the binary search algorithm.
Collections.copy(List dest, List src)	Copy the source List into dest List.
Collections.frequency(Collection c, Object o)	Returns the number of elements in the specified collection class c (which implements Collection interface can be List, Set or Queue) equal to the specified object.
Collections.synchronizedCollection(Collection c)	Returns a synchronized (thread-safe) collection backed by the specified collection.

Arrays utility class

The Arrays class in java.util package is a part of the Java Collection Framework. This class provides static methods to dynamically create and access Java arrays. It consists of only static methods and the methods of Object class. The methods of this class can be used by the class name itself.

Syntax

```
public static <T> List<T> asList(T... a)

public static void sort(int[] a)

public static int binarySearch(int[] a, int k)

public static boolean equals(int[] a, int[] a2)
```

Java Arrays as List

Java Arrays provides asList method that returns a list backed by the specified array. Below is a simple program for showing array as a List.

Exercise 6: Implement ArrayAsList

In this exercise you will understand the implementation of MyArraysAsList.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyArraysAsListClass**.

```
// Java code to illustrate the use of iterator
import java.io.*;
import java.util.Arrays;
import java.util.List;

public class MyArraysAsListClass {

        public static void main(String[] args) {
                String[] strings = {"one", "two", "three", "four", "five"};
                // strings array is converted into a List
                List<String> list = Arrays.asList(strings);
                System.out.println(list);
        }
}
```

5. Click **Run** to view the output.

```
[one, two, three, four, five]
```

Java Arrays Sort

Java Arrays provides sort method that sorts the element of specified array and sorts the specified range of the given array into ascending order. Below is a simple program for sorting arrays in java.

Exercise 7: Implement Arrays Sort

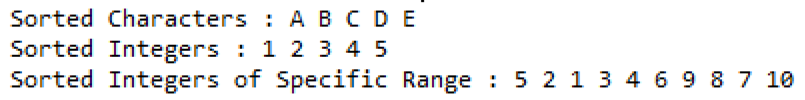
In this exercise you will understand the implementation of arrays sort.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyArraysSortClass**.

```
import java.util.*;
public class MyArraysSortClass{
        public static void main(String[] args) {
                // Sorting Character
                char[] chars = {'B', 'D', 'C', 'A', 'E'};
                Arrays.sort(chars);
                System.out.print("Sorted Characters : ");
                for (char character : chars) {
                        System.out.print(character+" ");
                }
        // Sorting Integer
                int[] integers = {5, 2, 1, 4, 3};
                Arrays.sort(integers);
                System.out.print("\nSorted Integers : ");
                for (int i : integers) {
                        System.out.print(i+" ");
                }
                // Sorting Specific Range of Integers
                int[] ints = {5, 2, 1, 4, 3, 9, 6, 8, 7, 10};
                int fromIndex = 2;
                int toIndex = 7;
                Arrays.sort(ints, fromIndex, toIndex);
                System.out.print("\nSorted Integers of Specific Range : ");
                for (int i : ints) {
                        System.out.print(i+" ");
                }
        }
}
```

5. Click **Run** to view the output.



```
Sorted Characters : A B C D E
Sorted Integers : 1 2 3 4 5
Sorted Integers of Specific Range : 5 2 1 3 4 6 9 8 7 10
```

Set

The Set interface and common implementations

Following are the features of set interface:

- Set is an interface which extends Collection. It is an unordered collection of objects in which duplicate values cannot be stored.
- Basically, Set is implemented by HashSet, LinkedHashSet or TreeSet (sorted representation).
- Set has various methods to add, remove clear, size, etc to enhance the usage of this interface

Exercise 8: Implement Set

In this exercise you will understand the implementation of set interface.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MySetClass**.

```
import java.util.Arrays;
import java.util.*;
import java.util.List;
import java.util.stream.Collectors;

public class MySetClass {

                              public static void main(String[] args) {
                                Set<String> hash_Set = new HashSet<String>();
                                hash_Set.add("This");
                                hash_Set.add("is");
                                hash_Set.add("Example");
```

```
                                    System.out.println(hash_Set);
                                  }
}
```

5. Click **Run** to view the output.

Now we will see some of the basic operations on the Set i.e. Union, Intersection and Difference.

Let's take an example of two integer Sets:

- [1, 3, 2, 4, 8, 9, 0]
- [1, 3, 7, 5, 4, 0, 7, 5]

Union

In this, we could simply add one Set with other. Since the Set will itself not allow any duplicate entries, we need not take care of the common values.

Intersection

We just need to retain the common values from both Sets.

Difference

We just need to remove all the values of one Set from the other.

Map

The Map interface and common implementations

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. This interface takes the place of the Dictionary class, which was a totally abstract class rather than an interface.

The Map interface provides three collection views, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings. The orderof a map is defined as the order in which the iterators on the map's collection views return their elements. Some map implementations, like the TreeMap class, make specific guarantees as to their order; others, like the HashMap class, do not.

All general-purpose map implementation classes should provide two "standard" constructors: a void (no arguments) constructor which creates an empty map, and a constructor with a single argument of type Map, which creates a new map with the same key-value mappings as its argument.

Exercise 9: Implement ArrayAsList

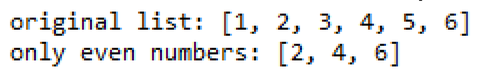
In this exercise you will understand the implementation of MyArraysAsList.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyMapClass**.

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
public class MyMapClass {
public static void main(String[] args) {
    List<String> numbers = Arrays.asList("1", "2", "3", "4", "5", "6");
    System.out.println("original list: " + numbers);
    List<Integer> even = numbers.stream().map(s ->
Integer.valueOf(s)).filter(number -> number % 2 == 0)
                                          .collect(Collectors.toList());
    System.out.println("only even numbers: " + even);
                         }
}
```

5. Click **Run** to view the output.



```
original list: [1, 2, 3, 4, 5, 6]
only even numbers: [2, 4, 6]
```

Implementations

Map implementations are grouped into general-purpose, special-purpose, and concurrent implementations.

General-Purpose Map Implementations

The three general-purpose Map implementations are HashMap, TreeMap and LinkedHashMap.

- If you need SortedMap operations or key-ordered Collection-view iteration, use TreeMap.
- if you want maximum speed and don't care about iteration order, use HashMap.
- if you want near-HashMap performance and insertion-order iteration, use LinkedHashMap.

Special-Purpose Map Implementations

There are three special-purpose Map implementations — EnumMap, WeakHashMap and IdentityHashMap.

- EnumMap, which is internally implemented as an array, is a high-performance Map implementation for use with enum keys. This implementation combines the richness and safety of the Map interface with a speed approaching that of an array. If you want to map an enum to a value, you should always use an EnumMap in preference to an array.

- WeakHashMap is an implementation of the Map interface that stores only weak references to its keys. Storing only weak references allows a key-value pair to be garbage-collected when its key is no longer referenced outside of the WeakHashMap.
- IdentityHashMap is an identity-based Map implementation based on a hash table. This class is useful for topology-preserving object graph transformations, such as serialization or deep-copying. To perform such transformations, you need to maintain an identity-based "node table" that keeps track of which objects have already been seen.

Concurrent Map Implementations

ConcurrentHashMap is a highly concurrent, high-performance implementation backed up by a hash table. This implementation never blocks when performing retrievals and allows the client to select the concurrency level for updates.

Queue

The Queue interface and common implementations

The Queue interface is available in java.util package and extends the Collection interface. The queue collection is used to hold the elements about to be processed and provides various operations like the insertion, removal etc. It is an ordered list of objects.

Few important characteristics of Queue are:

- The Queue is used to insert elements at the end of the queue and removes from the beginning of the queue. It follows FIFO concept.
- The Java Queue supports all methods of Collection interface including insertion, deletion etc.

- LinkedList, ArrayBlockingQueue and PriorityQueue are the most frequently used implementations.
- If any null operation is performed on BlockingQueues, NullPointerException is thrown.
- BlockingQueues have thread-safe implementations.
- The Queues which are available in java.util package are Unbounded Queues
- The Queues which are available in java.util.concurrent package are the Bounded Queues.
- All Queues except the Deques supports insertion and removal at the tail and head of the queue respectively. The Deques support element insertion and removal at both ends.

Exercise 10: Implement Queue

In this exercise you will understand the implementation of queue.

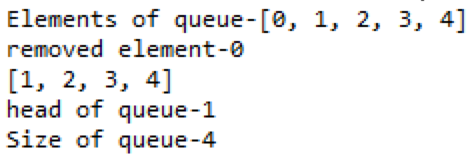
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyQueueClass**.

```
import java.util.*;

public class MyQueueClass{
public static void main(String[] args) {
                    Queue<Integer> q = new LinkedList<>();

                    // Adds elements {0, 1, 2, 3, 4} to queue
                    for (int i=0; i<5; i++)
                    q.add(i);
```

```
                        // Display contents of the queue.
                        System.out.println("Elements of queue-"+q);

                        // To remove the head of queue.
                        int removedele = q.remove();
                        System.out.println("removed element-" +
removedele);

                        System.out.println(q);

                        // To view the head of queue
                        int head = q.peek();
                        System.out.println("head of queue-" + head);

                        // Rest all methods of collection interface,
                        // Like size and contains can be used with this
                        // implementation.
                        int size = q.size();
                        System.out.println("Size of queue-" + size);

                }
}
```

5. Click **Run** to view the output.



```
Elements of queue-[0, 1, 2, 3, 4]
removed element-0
[1, 2, 3, 4]
head of queue-1
Size of queue-4
```

The Queue implementations are grouped into general-purpose and concurrent implementations.

General-Purpose Queue Implementations

LinkedList implements the Queue interface, providing first in, first out (FIFO) queue operations for add, poll, and so on.

The PriorityQueue class is a priority queue based on the heap data structure. This queue orders elements according to the order specified at construction time, which can be the elements' natural ordering, or the ordering imposed by an explicit Comparator.

The queue retrieval operations — poll, remove, peek, and element — access the element at the head of the queue. The head of the queue is the least element with respect to the specified ordering. If multiple elements are tied for least value, the head is one of those elements; ties are broken arbitrarily.

Concurrent Queue Implementations

The java.util.concurrent package contains a set of synchronized Queue interfaces and classes. BlockingQueue extends Queue with operations that wait for the queue to become nonempty when retrieving an element and for space to become available in the queue when storing an element. This interface is implemented by the following classes:

Queue	Description
LinkedBlockingQueue	an optionally bounded FIFO blocking queue backed by linked nodes
ArrayBlockingQueue	a bounded FIFO blocking queue backed by an array
PriorityBlockingQueue	an unbounded blocking priority queue backed by a heap

SynchronousQueue	a simple rendezvous mechanism that uses the BlockingQueue interface.
DelayQueue	a time-based scheduling queue backed by a heap

Legacy Collections

Hashtable and Properties

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

- A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
- Java Hashtable class contains unique elements.
- Java Hashtable class doesn't allow null key or value.
- Java Hashtable class is synchronized.
- The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.

```
public class Hashtable<K,V> extends Dictionary<K,V> implements Map<
K,V>, Cloneable, Serializable
```

Exercise 11: Implement Hashtable

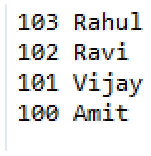
In this exercise you will understand the implementation of hashtable.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyHashtableClass**.

```
import java.util.*;
public class MyHashtableClass{
                          public static void main(String args[]){
                           Hashtable<Integer,String> hm=new
Hashtable<Integer,String>();
                           hm.put(100,"Amit");
                           hm.put(102,"Ravi");
                           hm.put(101,"Vijay");
                           hm.put(103,"Rahul");
                           for(Map.Entry m:hm.entrySet()){
                             System.out.println(m.getKey()+"
"+m.getValue());
                           }
                          }
}
```

5. Click **Run** to view the output.



```
103 Rahul
102 Ravi
101 Vijay
100 Amit
```

Reading from properties file using Properties class

Dot [.]properties is a file extension for files mainly used in Java related technologies to store the configurable parameters of an application. They can also be used for storing strings for Internationalization and localization; these are known as Property Resource Bundles.

Syntax

```
Properties appProps = new Properties();
appProps.load(new FileInputStream(Path));

String appVersion = appProps.getProperty("version");
```

Comparing Objects

- Java Comparable interface is used to order the objects of the user-defined class.
- This interface is found in java.lang package and contains only one method named compareTo(Object).

Comparable objects

- A comparable object is capable of comparing itself with another object.
- The class itself must implements the **java.lang.Comparable** interface to compare its instances.

Working with comparable objects

The Comparable interface has **compareTo(T obj)** method which is used by sorting methods, you can check any Wrapper, String or Date class to confirm this. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if “this” object is less than, equal to, or greater than the object passed as an argument.

Exercise 12: Implement Compareto

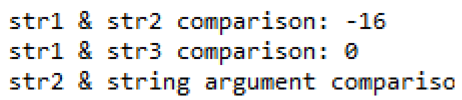
In this exercise you will understand the implementation of compareto using String class.

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.

```
public class MyCompareToClass{
  public static void main(String args[]) {
     String str1 = "String testing";
     String str2 = "compareTo method example";
     String str3 = "String testing";

     int var1 = str1.compareTo( str2 );
     System.out.println("str1 & str2 comparison: "+var1);

     int var2 = str1.compareTo( str3 );
     System.out.println("str1 & str3 comparison: "+var2);

     int var3 = str2.compareTo("compareTo method example");
     System.out.println("str2 & string argument comparison: "+var3);
  }
}
```

4. Click **Run** to view the output.



```
str1 & str2 comparison: -16
str1 & str3 comparison: 0
str2 & string argument compariso
```

Comparator

Comparator is external to the element type we are comparing. It's a separate class. We create multiple separate classes (that implement Comparator) to compare by different members.

Collections class has a second sort() method and it takes Comparator. The sort() method invokes the compare() to sort objects.

To compare movies by Rating, we need to do 3 things:

1. Create a class that implements Comparator (and thus the compare() method that does the work previously done by compareTo()).
2. Make an instance of the Comparator class.
3. Call the overloaded sort() method, giving it both the list and the instance of the class that implements Comparator.

To summarize, if sorting of objects needs to be based on natural order then use Comparable whereas if you are sorting needs to be done on attributes of different objects, then use Comparator in Java.

Test Your Knowledge

1. What is the output of the following program?

```
import java.util.*;
public class priorityQueue {
    public static void main(String[] args)  {
        PriorityQueue<Integer> queue =  new PriorityQueue<>();
        queue.add(11);
        queue.add(10);
        queue.add(22);
        queue.add(5);
        queue.add(12);
        queue.add(2);
          while (queue.isEmpty() == false)
          System.out.printf("%d ", queue.remove());
          System.out.println("\n");
    }
}
```

A. 11 10 22 5 12 2
B. 2 12 5 22 10 11
C. 2 5 10 11 12 22
D. 22 12 11 10 5 2

2. Which class does not override the equals() and hashCode() methods, inheriting them directly from class Object?

A. java.lang.String
B. java.lang.Double
C. java.lang.StringBuffer
D. java.lang.Character

3. You need to store elements in a collection that guarantees that no duplicates are stored, and all elements can be accessed in natural order. Which interface provides that capability?

   A. java.util.Map
   B. java.util.Set
   C. java.util.List
   D. java.util.Collection

4. Which interface does java.util.Hashtable implement?

   A. Java.util.Map
   B. Java.util.List
   C. Java.util.HashTable
   D. Java.util.Collection

5. Which collection class allows you to associate its elements with key values, and allows you to retrieve objects in FIFO (first-in, first-out) sequence?

   A. java.util.ArrayList
   B. java.util.LinkedHashMap
   C. java.util.HashMap
   D. java.util.TreeMap

Chapter 11: Lambda Expressoins

Basic of lambda expression

What is lambda expression ?

Lambda expressions provide a different way to represent method using an expression. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

Exercise 1: Implement Lambda

In this exercise you will understand the implementation of lambda .

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyLambdaImplClass**.

```
//Functional interface (interface with single abstract method)
interface Lambda {
        // abstract function
        void abstractFun(int x);
}
public class MyLambdaImplClass {
        public static void main(String args[]) {
                //lambda expression to implement functional interface
                Lambda lobj = (int x) -> System.out.println(5 * x);
                lobj.abstractFun(5);
        }
}
```

5. Click **Run** to view the output.

```
25
```

Java Training Module

Need for Lambda Expression

It is very useful in collection library , it helps to iterate , filter and extract data from collection.

It provides below functionalities

1. Enable to treat functionality as a method argument, or code as data.
2. A function that can be created without belonging to any class.
3. A lambda expression can be passed around as if it was an object and executed on demand

Syntax of Lambda expression

```
lambda operator -> body
```

Sample code

```
() -> System.out.println("Lambdas syntax");
```

```
() -> {
   double pi = 3.1415;
   return pi;
}
```

Functional Interface and Implementation

An interface with exactly with one abstract method is called functional interface. lambda expressions can be used to represent the instance of a functional interface.

An important point to remember is that the functional interface can have a few default methods but only one abstract method.

Implementation

A Java lambda expression implements a single method from a Java interface. In order to know what method, the lambda expression implements, the interface can only contain a single unimplemented method. In other words, the interface must be a Java functional interface.

Type Inference

One lambda expression may map to different functional interface types depending on the context. The compiler infers the type of a lambda expression

Functional interfaces in Java 8

An Interface that contains exactly one abstract method is known as functional interface. It can have any number of defaults, static methods but can contain only one abstract method.

Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve functional programming approach.

Using Functional Interfaces

Functional interface can be use with @Functional annotation with the functional interface.

Sorting List using Lambda Expression

Exercise 2: Implement Lambda using collection framework

In this exercise you will understand the implementation of lambda using collections framework.

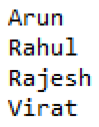
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyLambdaDemo**.

```
import java.util.*;
class PersonName {
	String fname;
	String lname;

	public PersonName(String fname, String lname) {
		super();
		this.fname = fname;
		this.lname = lname;
	}

}
public class MyLambdaDemo {

	public static void main(String args[]) {
		// define list
		List<PersonName> personList = new ArrayList<>();
		personList.add(new PersonName("Virat", "Kohli"));
		personList.add(new PersonName("Arun", "Kumar"));
		personList.add(new PersonName("Rajesh", "Mohan"));
		personList.add(new PersonName("Rahul", "Dravid"));
```

```
                // sorting
                Collections.sort(personList, new
Comparator<PersonName>() {
                        public int compare(PersonName p1, PersonName p2)
{
                                return p1.fname.compareTo(p2.fname);
                        }
                });

                // print using foreach
                personList.forEach((s) -> System.out.println(s.fname));
        }
}
```

5. Click **Run** to view the output.



```
Arun
Rahul
Rajesh
Virat
```

Test Your Knowledge

1. Given

```
interface Test {
 public void print( );
}
```

Which are valid lambda expressions (select 2 options)?

A. ->System.out.println(“Hello world”);
B. void -> System.out.println(“Hello world”);
C. ( ) -> System.out.println(“Hello world”);
D. ( ) ->{ System.out.println(“Hello world”); return; }
E. (void) -> System.out.println(“Hello world”);

2. Which are true about the functional interface?

A. It has exactly one method and it must be abstract.
B. It has exactly one method and it may or may not be abstract.
C. It must have exactly one abstract method and may have any number of default or static methods.
D. It must have exactly one default method and may have any number of abstract or static methods.
E. It must have exactly one static method and may have any number of default or abstract methods.

Java Training Module

3. Which of the following functional interface represents a function that accepts an int-valued argument and produces a double-valued result?

A. IntToDoubleFunction
B. IntToLongFunction
C. IntUnaryOperator
D. LongBinaryOperator

4. How will you call a static method of an interface in a class?

A. Using super keyword along with interface name.
B. Using name of the interface.
C. Both of the above.
D. None of the above.

5. Which of the following is correct about Java 8 lambda expression?

A. Using lambda expression, you can refer to final variable or effectively final variable (which is assigned only once).
B. Lambda expression throws a compilation error, if a variable is assigned a value the second time.
C. Both of the above.
D. None of the above.

Chapter 12:JDBC

Overview of JDBC API

JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

JDBC helps you to write Java applications that manage these three programming activities:

1. Connect to a data source, like a database
2. Send queries and update statements to the database
3. Retrieve and process the results received from the database in answer to your query

Types of driver

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver
3. Type-3 driver or Network Protocol driver
4. Type-4 driver or Thin driver

Type-1 driver

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.

- As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.

- The ODBC bridge driver is needed to be installed in individual client machines.
- Type-1 driver isn't written in java, that's why it isn't a portable driver.

Type-2 driver

The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.

- Driver needs to be installed separately in individual client machines
- The Vendor client library needs to be installed on client machine.
- Type-2 driver isn't written in java, that's why it isn't a portable driver

Type-3 driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation.

- Type-3 drivers are fully written in Java; hence they are portable drivers.
- No client-side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- Network support is required on client machine.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

Type-4 driver

Type-4 driver is also called native protocol driver. This driver interacts directly with database. It does not require any native database library, that is why it is also known as Thin Driver.

- Does not require any native library and Middleware server, so no client-side or server-side installation.
- It is fully written in Java language; hence they are portable drivers.

Which Driver to use When?

- If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is type-4.
- If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
- The type 1 driver is not considered a deployment-level driver and is typically used for development and testing purposes only.

JDBC interface and classes for connecting and retrieving data

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

The details about each step are:

1) Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

public static void forName(String className)throws ClassNotFoundException

Example:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2) Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

- public static Connection getConnection(String url)throws SQLException
- public static Connection getConnection(String url,String name,String password) throws SQLException

Example

```
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

public Statement createStatement()throws SQLException

Example

```
Statement stmt=con.createStatement();
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

public ResultSet executeQuery(String sql)throws SQLException

Example

```
ResultSet rs=stmt.executeQuery("select * from emp");
    while(rs.next()){
        System.out.println(rs.getInt(1)+" "+rs.getString(2));  }
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

public void close()throws SQLException

Example

```
con.close();
```

Exercise 1: Implement JDBC

In this exercise you will understand the implementation of insert a record in a table.

As part of the real-time, the tables will be created at the back end. In this exercise, oracle database instructions are provided in the Java code to insert the record. You can also use database such as MYSQL, DB2 to connect with java application.

Before we start developing the code, you need get database installed, and get following details. The details provided here should be replaced with your database credentials:

- IP address of the database system: 9.204.168.69
- Port number of the database running: 1521
- Schema/Service name: orcl
- Username: ibm
- Password: P#ssw0rd

In addition to the credentials, jdbc.jar file needs to place in the eclipse package to access the database

At the database level, for this example, the table is created using following syntax.

```
SQL> CREATE TABLE userid(id varchar2(30) NOT NULL PRIMARY
KEY,pwd varchar2(30) NOT NULL,fullname varchar2(50),email
varchar2(50));
```

Procedure

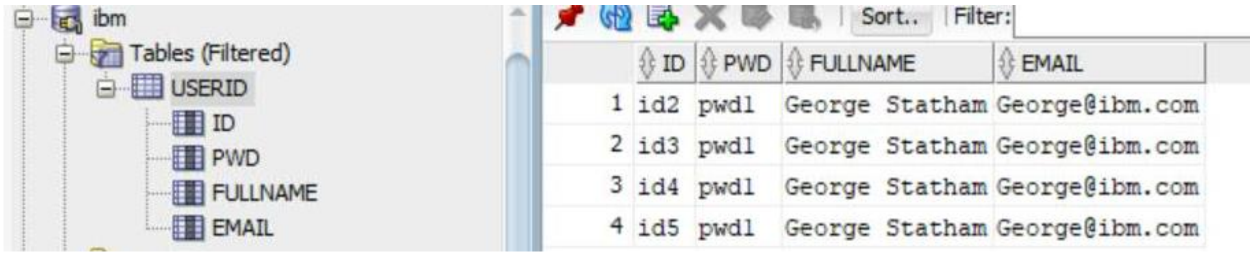
1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MyDBCheckClass**.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class MyDBCheckClass {
public static void main(String[] args) {
try
{
String id = "id2";
String pwd = "pwd1";
String fullname = "George Statham";
String email = "George@ibm.com";
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con = DriverManager.getConnection(
"jdbc:oracle:thin:@9.204.168.69:1521:orcl", "ibm", "P#ssw0rd");

if (con != null)
System.out.println("Connected");
else
System.out.println("Not Connected");

// Inserting data in database
String q1 = "insert into userid values('" +id+ "', '" +pwd+ "', '"
+fullname+ "', '" +email+ "')";
Statement stmt = con.createStatement();
int x = stmt.executeUpdate(q1);
if (x > 0)
System.out.println("Successfully Inserted");
else
System.out.println("Insert Failed");
con.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

5. Click **Run** to view the output.

```
Connected
Successfully Inserted
```

Cross check the inserted record in the database. Here, SQLDeveloper tool is used see the table details. You can use any database tool to access the tables.



	ID	PWD	FULLNAME	EMAIL
1	id2	pwd1	George Statham	George@ibm.com
2	id3	pwd1	George Statham	George@ibm.com
3	id4	pwd1	George Statham	George@ibm.com
4	id5	pwd1	George Statham	George@ibm.com

Type of Statement

Once a connection is obtained, we can interact with the database. The JDBC *Statement, CallableStatement,* and *PreparedStatement* interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.

They also define methods that help bridge data type differences between Java and SQL data types used in a database.

There are three statements interfaces:

Interface	Description
Statement	Use the for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Use the when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use the when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

Creating and closing a PreparedStatement

The *PreparedStatement* interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object.

This statement gives you the flexibility of supplying arguments dynamically.

Example

```
PreparedStatement pstmt = null;
try {
  String SQL = "Update Employees SET age = ? WHERE id = ?";
  pstmt = conn.prepareStatement(SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  pstmt.close();

}
```

Exercise 2: Implement PreparedStatement

In this exercise you will understand the implementation of fetch a record using unique ID field.

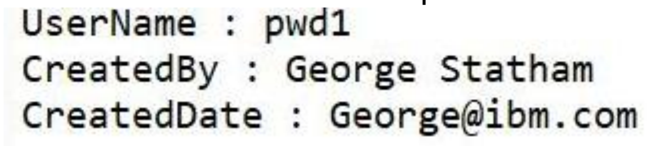
Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MydbCheckPreparedStatement**.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

public class MydbCheckPreparedStatement {

	public static void main(String[] args) {
		try {
			Class.forName("oracle.jdbc.driver.OracleDriver");

			Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@9.204.168.69:1521:
orcl", "ibm", "P#ssw0rd");

			if (con != null)
				System.out.println("Connected");
			else
				System.out.println("Not Connected");

			PreparedStatement pstmt = null;
```

```
                ResultSet rs = null;

                String query = "select * from userid where id = ?";
                pstmt = con.prepareStatement(query);
                pstmt.setString(1, "id2");
                rs = pstmt.executeQuery();
                if (rs.next()) {
                    System.out.println(rs.getString(1));
                }
                con.close();
            } catch (Exception e) {
                System.out.println(e);
            }
        }
}
```

5. Click **Run** to view the output.



```
UserName : pwd1
CreatedBy : George Statham
CreatedDate : George@ibm.com
```

Creating and closing a CallableStatement

Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object, which would be used to execute a call to a database stored procedure.

Example

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   cstmt.close();
}
```

ResultSet Interface and Its Methods

A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

A ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set.

The following code fragment, in which con is a valid Connection object, illustrates how to make a result set that is scrollable and insensitive to updates by others, and that is updatable. See ResultSet fields for other options.

Example

```
Statement stmt = con.createStatement(
                          ResultSet.TYPE_SCROLL_INSENSITIVE,
                          ResultSet.CONCUR_UPDATABLE);
     ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");
     // rs will be scrollable, will not show changes made by others,

     // and will be updatable
```

Commonly used Methods of ResultSet:

Following are few methods of resultset:

Interface	Description
public boolean next():	used to move the cursor to the one row next from the current position.
public boolean previous():	used to move the cursor to the one row previous from the current position.
public boolean first():	used to move the cursor to the first row in result set object.
public boolean last()	used to move the cursor to the last row in result set object.
boolean absolute(int row)	used to move the cursor to the specified row number in the ResultSet object.

Batch Processing

Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast.

The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.

Example

Below is an example of batch processing

- Load the driver class
- Create Connection
- Create Statement
- Add query in the batch
- Execute Batch
- Close Connection

```
   import java.sql.*;
   class FetchRecords{
   public static void main(String args[])throws Exception{
   Class.forName("oracle.jdbc.driver.OracleDriver");
   Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:x
e","system","oracle");
   con.setAutoCommit(false);
   Statement stmt=con.createStatement();
   stmt.addBatch("insert into user420 values(190,'abhi',40000)");
   stmt.addBatch("insert into user420 values(191,'umesh',50000)");
   stmt.executeBatch();//executing the batch
    con.commit();
   con.close();

   }}
```

Transaction Management

There are times when you do not want one statement to take effect unless another one completes.

For example, when the proprietor of The Coffee Break updates the amount of coffee sold each week, the proprietor will also want to update the total amount sold to date. However, the amount sold per week and the total amount sold should be updated at the same time; otherwise, the data will be inconsistent.

The way to be sure that either both actions occur or neither action occurs is to use a transaction. A transaction is a set of one or more statements that is executed as a unit, so either all of the statements are executed, or none of the statements is executed.

Disabling Auto-Commit Mode

When a connection is created, it is in auto-commit mode. This means that each individual SQL statement is treated as a transaction and is automatically committed right after it is executed. (To be more precise, the default is for a SQL statement to be committed when it is completed, not when it is executed. A statement is completed when all of its result sets and update counts have been retrieved. In almost all cases, however, a statement is completed, and therefore committed, right after it is executed.)

The way to allow two or more statements to be grouped into a transaction is to disable the auto-commit mode. This is demonstrated in the following code, where con is an active connection:

```
con.setAutoCommit(false);
```

Committing Transactions

After the auto-commit mode is disabled, no SQL statements are committed until you call the method commit explicitly. All statements executed after the previous call to the method commit are included in the current transaction and committed together as a unit. The following method, CoffeesTable.updateCoffeeSales, in which con is an active connection, illustrates a transaction:

```
public void updateCoffeeSales(HashMap<String, Integer>
salesForWeek)
    throws SQLException {

    PreparedStatement updateSales = null;
    PreparedStatement updateTotal = null;

    String updateString =
        "update " + dbName + ".COFFEES " +
        "set SALES = ? where COF_NAME = ?";

    String updateStatement =
        "update " + dbName + ".COFFEES " +
        "set TOTAL = TOTAL + ? " +
        "where COF_NAME = ?";

    try {
        con.setAutoCommit(false);
        updateSales = con.prepareStatement(updateString);
        updateTotal = con.prepareStatement(updateStatement);

        for (Map.Entry<String, Integer> e : salesForWeek.entrySet()) {
            updateSales.setInt(1, e.getValue().intValue());
            updateSales.setString(2, e.getKey());
```

Java Training Module

```
            updateSales.executeUpdate();
            updateTotal.setInt(1, e.getValue().intValue());
            updateTotal.setString(2, e.getKey());
            updateTotal.executeUpdate();
            con.commit();
        }
    } catch (SQLException e ) {
        JDBCTutorialUtilities.printSQLException(e);
        if (con != null) {
            try {
                System.err.print("Transaction is being rolled back");
                con.rollback();
            } catch(SQLException excep) {
                JDBCTutorialUtilities.printSQLException(excep);
            }
        }
    } finally {
        if (updateSales != null) {
            updateSales.close();
        }
        if (updateTotal != null) {
            updateTotal.close();
        }
        con.setAutoCommit(true);
    }
```

In this method, the auto-commit mode is disabled for the connection con, which means that the two prepared statements updateSales and updateTotal are committed together when the method commit is called. Whenever the commit method is called (either automatically when auto-commit mode is enabled or explicitly when it is disabled), all changes resulting from statements in the transaction are made permanent. In this case, that means that the SALES and TOTAL columns for Colombian coffee have been changed to 50 (if TOTAL had been 0 previously) and will retain this value until they are changed with another update statement.

The statement con.setAutoCommit(true); enables auto-commit mode, which means that each statement is once again committed automatically when it is completed. Then, you are back to the default state where you do not have to call the method commit yourself. It is advisable to disable the auto-commit mode only during the transaction mode. This way, you avoid holding database locks for multiple statements, which increases the likelihood of conflicts with other users.

Transaction Rollback

The method Connection.setSavepoint, sets a Savepoint object within the current transaction. The Connection.rollback method is overloaded to take a Savepoint argument.

Retrieving Metadata/DatabaseMetadata

Database Metadata interface is generally implemented by the database vendor and provided along with the native JDBC driver. Native JDBC drivers are built on top of the database. By implementing this interface, database vendors provide comprehensive information about the database, such as table name, indexes, product name, version, and so forth. There are several methods declared in this interface to retrieve various metadata information associated with the database.

Example

```
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
 public class Metadata {
```

Java Training Module

```
    static Connection = null;
    static DatabaseMetaData metadata = null;
     // Static block for initialization
    static {
        try {
            connection = DBConnection.getConnection();
        } catch (SQLException e) {
            System.err.println("There was an error getting the connection: "
                    + e.getMessage());
        }
         try {
            metadata = connection.getMetaData();
        } catch (SQLException e) {
            System.err.println("There was an error getting the metadata: "
                    + e.getMessage());
        }
    }
     /**
     * Prints in the console the general metadata.
     *
     * @throws SQLException
     */
    public static void printGeneralMetadata() throws SQLException {
        System.out.println("Database Product Name: "
                + metadata.getDatabaseProductName());
        System.out.println("Database Product Version: "
                + metadata.getDatabaseProductVersion());
        System.out.println("Logged User: " + metadata.getUserName());
        System.out.println("JDBC Driver: " + metadata.getDriverName());
        System.out.println("Driver Version: " +
metadata.getDriverVersion());
        System.out.println("\n");
```

Java Training Module

```
}
 /**
 *
 * @return Arraylist with the table's name
 * @throws SQLException
 */
public static ArrayList getTablesMetadata() throws SQLException {
   String table[] = { "TABLE" };
   ResultSet rs = null;
   ArrayList tables = null;
   // receive the Type of the object in a String array.
   rs = metadata.getTables(null, null, null, table);
   tables = new ArrayList();
   while (rs.next()) {
      tables.add(rs.getString("TABLE_NAME"));
   }
   return tables;
}
 /**
 * Prints in the console the columns metadata, based in the Arraylist of
 * tables passed as parameter.
 *
 * @param tables
 * @throws SQLException
 */
public static void getColumnsMetadata(ArrayList tables)
      throws SQLException {
   ResultSet rs = null;
   // Print the columns properties of the actual table
   for (String actualTable : tables) {
      rs = metadata.getColumns(null, null, actualTable, null);
      System.out.println(actualTable.toUpperCase());
      while (rs.next()) {
            System.out.println(rs.getString("COLUMN_NAME") + " "
                  + rs.getString("TYPE_NAME") + " "
                  + rs.getString("COLUMN_SIZE"));
         }
         System.out.println("\n");
      }
   }
   /**
    *
    * @param args
    */
   public static void main(String[] args) {
      try {
         printGeneralMetadata();
         // Print all the tables of the database scheme, with their names
and
         // structure
         getColumnsMetadata(getTablesMetadata());
      } catch (SQLException e) {
         System.err
               .println("There was an error retrieving the metadata
properties: "
                     + e.getMessage());
      }
   }
}
```

In the above example, we just used two methods from the DatabaseMetaData interface, the first method getTables returns a Resultset object with the information about the type we send as a parameter, the typical types are “TABLE”, “VIEW”, “SYSTEM TABLE”, "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS", "SYNONYM". In the Resultset we can get the information looking for some columns as:

- TABLE_CAT String => table catalog (may be null)
- TABLE_SCHEM String => table schema (may be null)
- TABLE_NAME String => table name

With the second method getColumns we can obtain the information for each table, and with a forloop just get all the information from the database if we pass as a parameter the table name retrieved in the previous method.

RowSet and RowSet Types

A JDBC RowSet object holds tabular data in a way that makes it more flexible and easier to use than a result set.

Oracle has defined five RowSet interfaces for some of the more popular uses of a RowSet, and standard reference are available for these RowSet interfaces. In this tutorial you will learn how to use these reference implementations.

These versions of the RowSet interface and their implementations have been provided as a convenience for programmers. Programmers are free to write their own versions of the javax.sql.RowSet interface, to extend the implementations of the five RowSet interfaces, or to write their own implementations. However, many programmers will probably find that the standard reference implementations already fit their needs and will use them as is.

Connected and Disconnected RowSet

A RowSet object is considered either connected or disconnected. A connected RowSet object uses a JDBC driver to make a connection to a relational database and maintains that connection throughout its life span. A disconnected RowSet object makes a connection to a data source only to read in data from a ResultSet object or to write data back to the data source. After reading data from or writing data to its data source, the RowSet object disconnects from it, thus becoming "disconnected."

During much of its life span, a disconnected RowSet object has no connection to its data source and operates independently. The next two sections tell you what being connected or disconnected means in terms of what a RowSet object can do.

Connected RowSet Objects

Only one of the standard RowSet implementations is a connected RowSet object: JdbcRowSet. Always being connected to a database, a JdbcRowSet object is most similar to a ResultSet object and is often used as a wrapper to make an otherwise non-scrollable and read-only ResultSet object scrollable and updatable.

Disconnected RowSet Objects

The other four implementations are disconnected RowSet implementations. Disconnected RowSet objects have all the capabilities of connected RowSet objects plus they have the additional capabilities available only to disconnected RowSet objects. For example, not having to maintain a connection to a data source makes disconnected RowSet objects far more lightweight than a JdbcRowSet object or a ResultSet object. Disconnected RowSet objects are also serializable, and the combination of being both serializable and lightweight makes them ideal for sending data over a network. They can even be used for sending data to thin clients such as PDAs and mobile phones.

The CachedRowSet interface defines the basic capabilities available to all disconnected RowSet objects. The other three are extensions of the CachedRowSet interface, which provide more specialized capabilities. The following information shows how they are related:

A CachedRowSet object has all the capabilities of a JdbcRowSet object plus it can also do the following:

- Obtain a connection to a data source and execute a query
- Read the data from the resulting ResultSet object and populate itself with that data
- Manipulate data and make changes to data while it is disconnected
- Reconnect to the data source to write changes back to it
- Check for conflicts with the data source and resolve those conflicts

A WebRowSet object has all the capabilities of a CachedRowSet object plus it can also do the following:

- Write itself as an XML document
- Read an XML document that describes a WebRowSet object

A JoinRowSet object has all the capabilities of a WebRowSet object (and therefore also those of a CachedRowSet object) plus it can also do the following:

- Form the equivalent of a SQL JOIN without having to connect to a data source
- A FilteredRowSet object likewise has all the capabilities of a WebRowSet object (and therefore also a CachedRowSet object) plus it can also do the following:
- Apply filtering criteria so that only selected data is visible. This is equivalent to executing a query on a RowSet object without having to use a query language or connect to a data source.

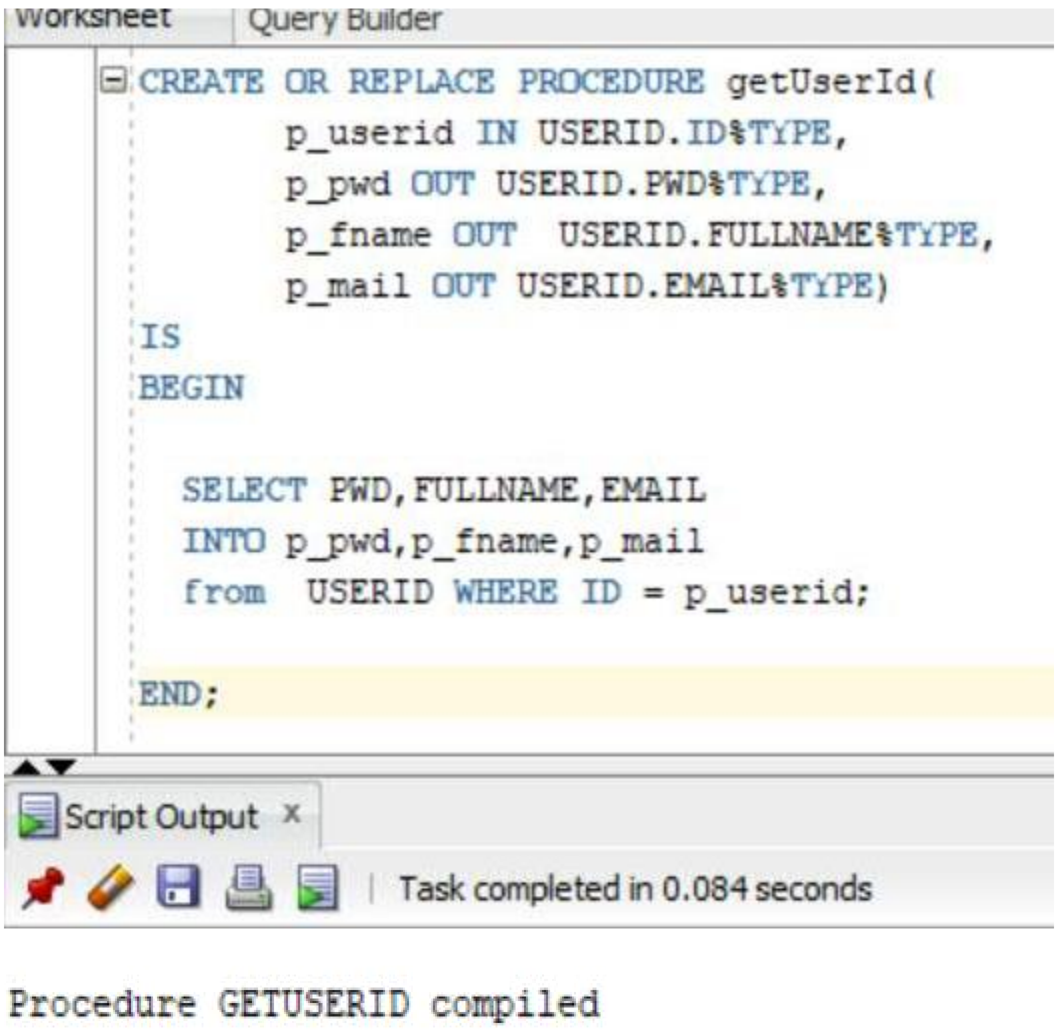
Exercise 3: Implement CallableStatement

In this exercise you will understand the implementation of fetch a record using unique ID field using the procedures.

Syntax of the procedure created in the database. Make sure that the procedure is working fine at the backend, before the running the callable statement from Java application.

```
CREATE OR REPLACE PROCEDURE
getUserId(
p_userid IN USERID.ID%TYPE,
p_pwd OUT USERID.PWD%TYPE,
p_fname OUT USERID.FULLNAME%TYPE,
p_mail OUT USERID.EMAIL%TYPE)
IS
BEGIN

SELECT PWD,FULLNAME,EMAIL
INTO p_pwd,p_fname,p_mail
from USERID WHERE ID = p_userid;

END;
```

Screen capture of the procedure getUserId created through SQLDeveloper tool.



```
CREATE OR REPLACE PROCEDURE getUserId(
        p_userid IN USERID.ID%TYPE,
        p_pwd OUT USERID.PWD%TYPE,
        p_fname OUT  USERID.FULLNAME%TYPE,
        p_mail OUT USERID.EMAIL%TYPE)
IS
BEGIN

  SELECT PWD,FULLNAME,EMAIL
  INTO p_pwd,p_fname,p_mail
  from  USERID WHERE ID = p_userid;

END;
```

```
Task completed in 0.084 seconds

Procedure GETUSERID compiled
```

Procedure

1. From the project you have created in the eclipse IDE tool.
2. Click **File** > **Project** > **Java** > **Java Project**
3. Click **Next**.
4. Project name as **MydbCheckPreparedStatement**.

```
import java.sql.CallableStatement;
import java.sql.Connection;
```

Java Training Module

```
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Types;

public class MyProceduralStatementClass {
	public static void main(String[] args) throws ClassNotFoundException,
SQLException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@9.204.168.69:1521:orcl",
"ibm", "P#ssw0rd");
		CallableStatement callableStatement = null;
		String getUserId = "{call getUserId(?,?,?,?)}";
		try {
			callableStatement = con.prepareCall(getUserId);
			//callableStatement.registerOutParameter(1,
Types.VARCHAR);
			callableStatement.setString(1, "id2");
			callableStatement.registerOutParameter(2,
java.sql.Types.VARCHAR);
			callableStatement.registerOutParameter(3,
java.sql.Types.VARCHAR);
			callableStatement.registerOutParameter(4,
java.sql.Types.VARCHAR);
			// execute getDBUSERByUserId store procedure
			callableStatement.executeUpdate();
			String pwd = callableStatement.getString(2);
			String fullname = callableStatement.getString(3);
			String mail = callableStatement.getString(4);
			System.out.println("UserName : " + pwd);
			System.out.println("CreatedBy : " + fullname);
			System.out.println("CreatedDate : " + mail);
		}catch (SQLException e) {
```

Java Training Module

```
                System.out.println(e.getMessage());
        } finally {
                if (callableStatement != null) {
                        callableStatement.close();
        }
                if (con != null) {
                        con.close();
                }
        }
}
```

5. Click **Run** to view the procedure execution through Java application.

Test Your Knowledge

1. Which of the following is advantage of using JDBC connection pool?

   A. Slow performance
   B. Using more memory
   C. Using less memory
   D. Better performance

2. Which of the following is advantage of using PreparedStatement in Java?

   A. Slow performance
   B. Encourages SQL injection
   C. Prevents SQL injection
   D. More memory usage

3. What does setAutoCommit(false) do?

   A. commits transaction after each query
   B. explicitly commits transaction
   C. does not commit transaction automatically after each query
   D. never commits transaction

4. Which of the following is method of JDBC batch process?

   A. setBatch()
   B. deleteBatch()
   C. removeBatch()
   D. addBatch()

Java Training Module

5. Which of the following is not a JDBC connection isolation levels?

   A. TRANSACTION_NONE
   B. TRANSACTION_READ_COMMITTED
   C. TRANSACTION_REPEATABLE_READ
   D. TRANSACTION_NONREPEATABLE_READ

Answer Keys

Chapter Name	Question Number -Answer Keys
Object Oriented concepts	1-C; 2-D; 3-C; 4-B; 5-C
Overview of Java Platform	1-D; 2-D; 3-D;4-A; 5-A
Java Language Fundamentals	1-A; 2-B; 3-A;4-A; 5-B
Creating Class and Objects	1-C; 2-C; 3-A;4-D; 5-C
Implementing OOP Concepts	1-A; 2-A; 3-A;4-C; 5-A
Useful Java API classes	1-B; 2-B; 3-D;4-D; 5-C
Exceptions	1-C; 2-B; 3-B;4-B; 5-A
File handling	1-B; 2-C; 3-A;4-C; 5-B
Multithreading	1-C; 2-D; 3-A;4-C; 5-C
Collection framework	1-C; 2-C; 3-B;4-A; 5-B
Lambda Expressions	1-C/D; 2-C; 3-A;4-B; 5-C
JDBC	1-D; 2-C; 3-C;4-D; 5-D