Introduction to Java Programming Language

Class

- A Class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. A class in Java can contain:
  - Fields
  - Methods
  - Constructors
  - Blocks
  - Nested class and interface

```
class Bicycle {

  // state or field
  private int gear = 5;

  // behavior or method
  public void braking() {
    System.out.println("Working of Braking");
  }
}
```

In the above example, we have created a class named `Bicycle`. It contains a field named `gear` and a method named `braking()`.

OBJECT

- An Object is called an instance of a class. For example, suppose Bicycle is a class then MountainBicycle, SportsBicycle, TouringBicycle, etc. can be considered as objects of the class. It is a basic unit of Object-Oriented Programming and represents the real life entities. An object consists of:
  - State: It is represented by attributes of an object. It also reflects the properties of an object.
  - Behavior: It is represented by methods of an object. It also reflects the response of an object with other objects.
  - Identity: It gives a unique name to an object and enables one object to interact with other objects.

Here is how we can create an object of a class.

```
className object = new className();

// for Bicycle class
Bicycle sportsBicycle = new Bicycle();

Bicycle touringBicycle = new Bicycle();
```

We have used the `new` keyword along with the constructor of the class to create an object. Constructors are similar to methods and have the same name as the class. For example,

Java programming Language

- Some buzzwords for Java
  - “Write Once, Run Anywhere”
  - Simple
  - **Object oriented**
  - Distributed
  - Multithreaded
  - Dynamic
  - **Architecture neutral**
  - Portable
  - High performance
  - Robust
  - Secure

Example: Hello World Program

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

- Public: It is an Access modifier, which specifies from where and who can access the method. Making the main() method public makes it globally available. It is made public so that JVM (Java Virtual Machine) can invoke it from outside the class as it is not present in the current class.
- Static: It is a keyword which is when associated with a method, makes it a class related method. The main() method is static so that JVM can invoke it without instantiating the class. This also saves the unnecessary wastage of memory which would have been used by the object declared only for calling the main() method by the JVM.

Example: Hello World Program

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

- Void: It is a keyword and used to specify that a method doesn't return anything. As main() method doesn't return anything, its return type is void. As soon as the main() method terminates, the java program terminates too. Hence, it doesn't make any sense to return from main() method as JVM can't do anything with the return value of it.
- Main: It is the name of Java main method. It is the identifier that the JVM looks for as the starting point of the java program. It's not a keyword.
- String[] args: It stores Java command line arguments and is an array of type java.lang.String class. Here, the name of the String array is args but it is not fixed and user can use any name in place of it.

Declaring Variables

```
int n = 1;
char ch = 'A';
String s = "Hello";
Long L = new Long(100000);
boolean done = false;
final double pi = 3.141592653589793846;
Employee joe = new Employee();
char [] a = new char[3];
Vector v = new Vector();
```

Object Oriented Programming (OOPs) Concept in Java

- As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.
- OOPs Concepts:
    - Inheritance
    - Polymorphism
    - Encapsulation
    - Abstraction

Inheritance in Java

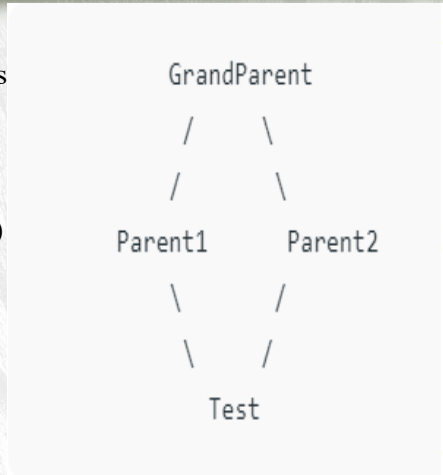
- Java classes can be *derived* from other classes, thereby *inheriting* fields and methods from those classes. It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.
  - ❖ Super Class: The class whose features are inherited is known as superclass(or a base class or a parent class).
  - ❖ Sub Class: The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
  - ❖ Reusability: Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Inheritance in Java

```
package inheritance;

public class Animal{
    private int age;
    public void move(){
        System.out.print("The Animal is moving");
    };
}

class Cat extends Animal{
    //a method in the sub-class
    public void meow(){
        System.out.print("The Cat is meowing");
    };
}

class Dog extends Animal{
    //a method in the sub-class
    public void bark(){
        System.out.print("The Dog is barking.");
    };
}
```

Why Java doesn’t support Multiple Inheritance?

- The Diamond Problem: - The problem with multiple inheritance is that two classes may define different ways of doing the same thing, and the subclass can't choose which one to pick.
- From the code, we see that: On calling the method fun() using Test object will cause complications such as whether to call Parent1’s fun() or Child’s fun() method.
- Therefore, in order to avoid such complications Java does not support multiple inheritance of classes.
- Simplicity: - Multiple inheritance is not supported by Java using classes , handling the complexity that causes due to multiple inheritance is very complex. It creates problem during various operations like casting, constructor chaining etc. and the above all reason is that there are very few scenarios on which we actually need multiple inheritance, so better to omit it for keeping the things simple and straightforward.



Made By Ankit Gupta

```
// A Grand parent class in diamond
class GrandParent
{
    void fun()
    {
        System.out.println("Grandparent");
    }
}
// First Parent class
class Parent1 extends GrandParent
{
    void fun()
    {
        System.out.println("Parent1");
    }
}
// Second Parent Class
class Parent2 extends GrandParent
{
    void fun()
    {
        System.out.println("Parent2");
    }
}
// Error : Test is inheriting from multiple
// classes
class Test extends Parent1, Parent2
{
   public static void main(String args[])
   {
       Test t = new Test();
       t.fun();
   }
}
```

Polymorphism in Java

- Polymorphism refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently. This is done by Java with the help of the signature and declaration of these entities. It allows us to perform a single action in different ways.
- Example :-

```
public class Sum {
    // This sum takes two int parameters
    public int sum(int x, int y)
    {
        return (x + y);
    }
    // This sum takes three int parameters
    public int sum(int x, int y, int z)
    {
        return (x + y + z);
    }
    // This sum takes two double parameters
    public double sum(double x, double y)
    {
        return (x + y);
    }
    public static void main(String args[])
    {
        Sum s = new Sum();
        System.out.println(s.sum(10, 20));
        System.out.println(s.sum(10, 20, 30));
        System.out.println(s.sum(10.5, 20.5));
    }
}
```

Encapsulation in Java

- Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.
  - Make the instance variables private so that they cannot be accessed directly from outside the class. You can only set and get values of these variables through the methods of the class.
  - Have getter and setter methods in the class to set and get the values of the fields.
- Example :-

```
class EmployeeCount
{
   private int numOfEmployees = 0;
   public void setNoOfEmployees (int count)
   {
       numOfEmployees = count;
   }
   public double getNoOfEmployees ()
   {
       return numOfEmployees;
   }
}
public class EncapsulationExample
{
   public static void main(String args[])
   {
      EmployeeCount obj = new EmployeeCount ();
      obj.setNoOfEmployees(5613);
      System.out.println("No Of Employees: "+(int)obj.getNoOfEmployees());
    }
}
```

Abstraction in Java

- Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.
- It may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.
- Example :-

```
//abstract class
abstract class Animal{
    //abstract method
    public abstract void animalSound();
}
public class Dog extends Animal{

    public void animalSound(){
          System.out.println("Woof");
    }
    public static void main(String args[]){
          Animal obj = new Dog();
          obj.animalSound();
    }
}
```

Interface in Java

- Like a class, an Interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).
  - Interfaces specify what a class must do and not how. It is the blueprint of the class.
  - An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
  - If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.
  - A Java library example is, Comparator Interface. If a class implements this interface, then it can be used to sort a collection.

Why do we use interface ?

- It is used to achieve total abstraction.
- Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .
- It is also used to achieve loose coupling.
- Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes?
- The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public and static.

Interface

```
package inheritance2;

public interface Animal {
    public void move();
    public void eat();
}

class Dog implements Animal
{
    public void move() {
        System.out.println("The Dog is moving.");
    }
    public void eat() {
        System.out.println("The Dog is eating.");
    }
}

class Cat implements Animal
{
    public void move() {
        System.out.println("The Dog is moving.");
    }
    public void eat() {
        System.out.println("The Dog is eating.");
    }
}
```

“Multiple Inheritance”

```
package inheritance2;

public interface Bird {
    public void fly();
}

interface MythologicalCreature{
    //Mythological Creatures can speak human languages
    public void speak();
}

class Horse {
    public void run(){
        System.out.println("The Horse is running");
    }
}

class Pegasus extends Horse implements Bird, MythologicalCreature{
    public void fly(){
        System.out.println("The Pegasus is running");
    }
    public void speak(){
        System.out.println("The Pegasus is speaking human languages");
    }
}
```

Basic Syntax

- About Java programs, it is very important to keep in mind the following points.
  - Case Sensitivity − Java is case sensitive, which means identifier Hello and hello would have different meaning in Java.
  - Class Names − For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

    Example: class MyFirstJavaClass
  - Method Names − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

    Example: public void myMethodName()
  - Program File Name − Name of the program file should exactly match the class name.

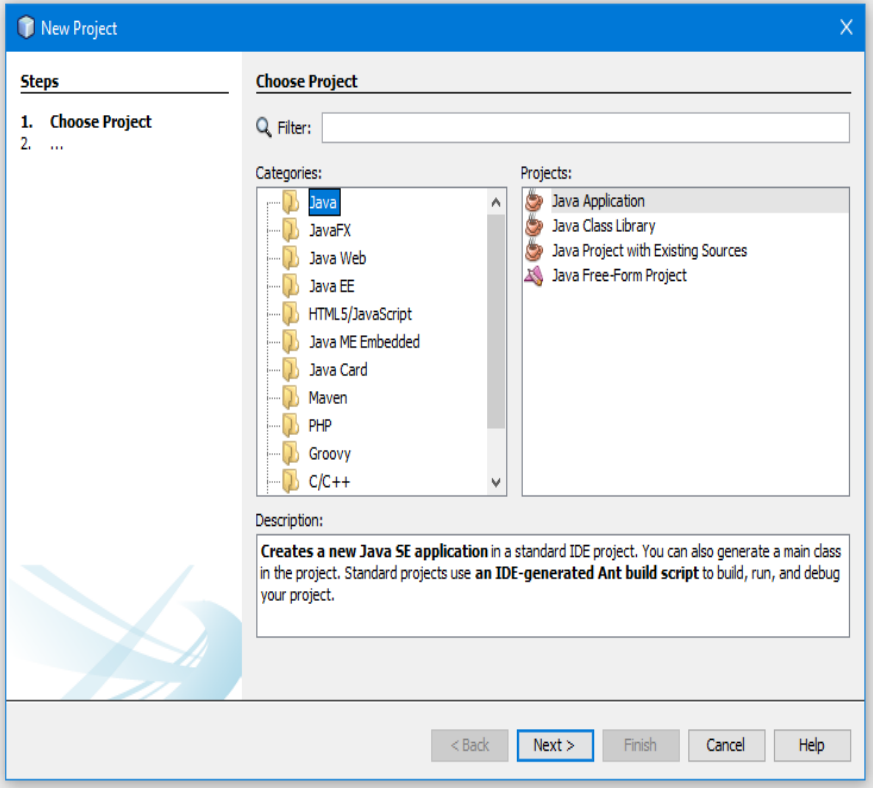
    But please make a note that in case you do not have a public class present in the file then file name can be different than class name. It is also not mandatory to have a public class in the file.

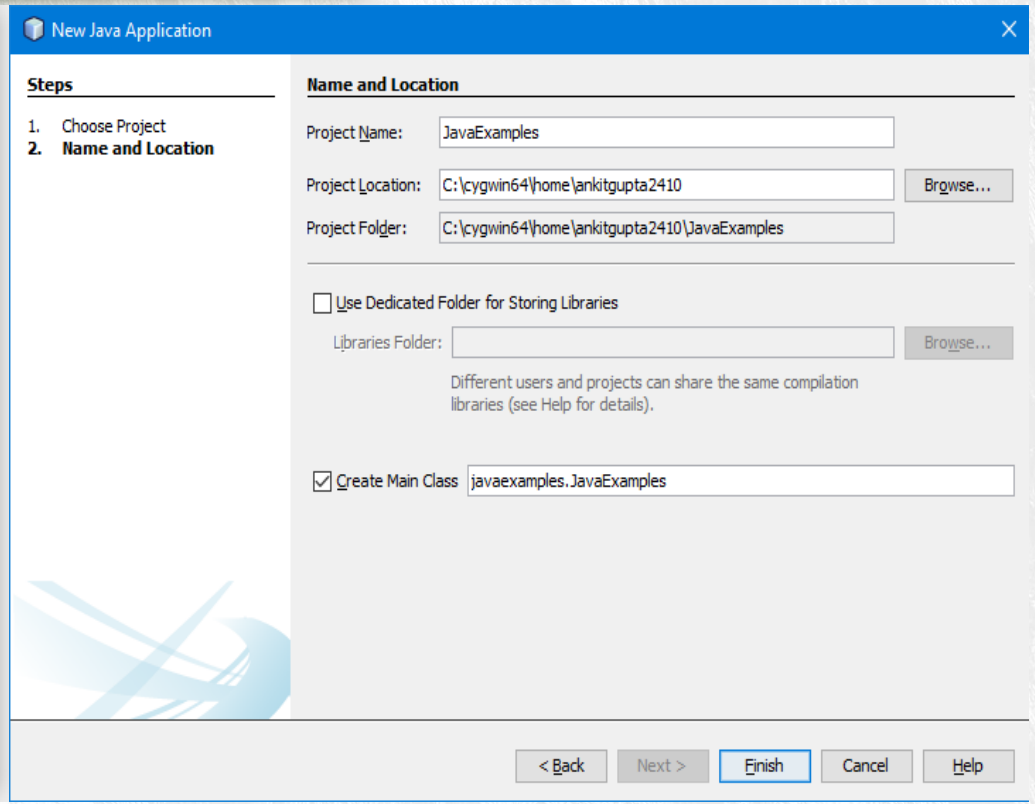
    Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as 'MyFirstJavaProgram.java'
  - public static void main(String args[]) − Java program processing starts from the main() method which is a mandatory part of every Java program.

Create A New Project In NetBeans

- Click on “File” from the Menu Option and then click on “New Project” or You can open it by shortcut :- “Ctrl + Shift + N”.
- Then Select “Java” from Categories and “Java Application” from Projects or select another according to the Project Need. After this, Click on “Next”.
- In Third Step give your “Project Name” and “Project Location” and Click on Check Box Button if you want to create “Main Class” Automatically or uncheck it ,if you want to create it manually later. Then Click on “Finish” Button for finishing the creation of the project.

Create A New Project In NetBeans





Starting Program

```
12  public class JavaExamples
13  {
14      public static void main(String[] args) {
15  //          If you want to give line break at the end of the Print statement:-
16  //          Then we have to use println instead of print which is used to print in the same line.
17          System.out.println("Welcome ");
18          System.out.print("to the ");
19          System.out.println("Java Application");
20      }
21  }
22
```

Output - JavaExamples (run)

```
run:
Welcome
to the Java Application
BUILD SUCCESSFUL (total time: 0 seconds)
```

User Input – Scanner Class

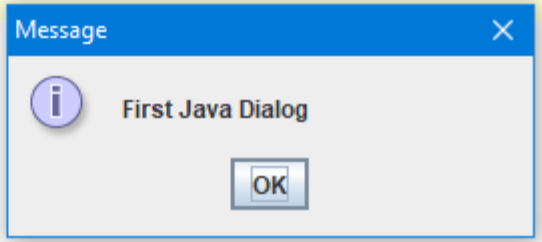
```
12  //Scanner is a class in java.util package used for obtaining the input of the primitive types
13  //like int, double, etc. and strings.
14  import java.util.Scanner;
15  public class UserInput {
16      public static void main(String[] args)
17      {
18          String name;
19          int age;
20  //          A Scanner object named inputDevice is declared.
21          Scanner inputDevice = new Scanner(System.in);
22          System.out.print("Please enter your name >> ");
23          /* The nextLine() method is used with inputDevice to retrieve a line of text
24          from the keyboard and store it in the name variable.*/
25          name = inputDevice.nextLine();
26          System.out.print("Please enter your age >> ");
27          /* The nextInt() method is used with inputDevice to retrieve an integer
28          from the keyboard and store it in the age variable.*/
29          age = inputDevice.nextInt();
30          System.out.println("Your name is " + name + " and you are " + age + " years old.");
31      }
32  }
33
```

Output - JavaExamples (run)

```
run:
Please enter your name >> Ankit
Please enter your age >> 10
Your name is Ankit and you are 10 years old.
BUILD SUCCESSFUL (total time: 8 seconds)
```

First Dialog in Java

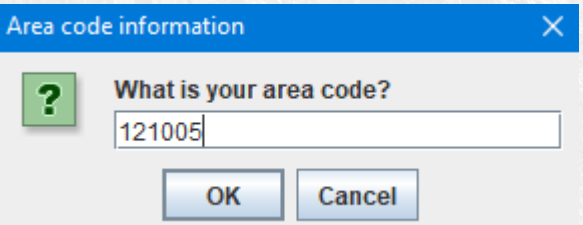
```
12  //For importing whole swing library module ,this code is used :-
13  //import javax.swing.*;
14  //For importing particular swing library module ,this code is used :-
15  import javax.swing.JOptionPane;
16  public class FirstDialog
17  {
18      public static void main(String[] args)
19      {
20          JOptionPane.showMessageDialog(null, "First Java Dialog");
21  //          Used for Older Verison of Jdk for Destroying Applet Programs.
22  //          For Newer Jdk Versions ,it is not required
23  //          System.exit(0);
24      }
25  }
```



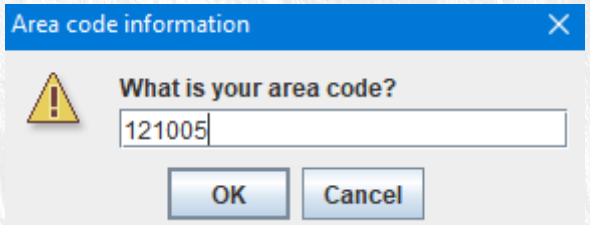
User Input Dialog And Type of Dialog Box

```
import javax.swing.JOptionPane;
public class UserNameDialog {
    public static void main(String[] args)
    {
        String result,result1;
        /*An input dialog box asks a question and provides a text field in which the user can enter a response.
        You can create an input dialog box using the showInputDialog() method.
        The showInputDialog() method returns a String that represents a user's response
        */
        result = JOptionPane.showInputDialog(null, "What is your name?");
        JOptionPane.showMessageDialog(null, "Hello, " + result + "!");

        /*A class field describing the type of dialog box can be described in InputDialog. It can be one of the following:
        ERROR_MESSAGE, INFORMATION_MESSAGE, PLAIN_MESSAGE, QUESTION_MESSAGE or WARNING_MESSAGE.*/

        result1 = JOptionPane.showInputDialog(null,"What is your area code?","Area code information",
                JOptionPane.INFORMATION_MESSAGE);
        JOptionPane.showMessageDialog(null, "Hello, " + result1 + "!");
 }
}
```

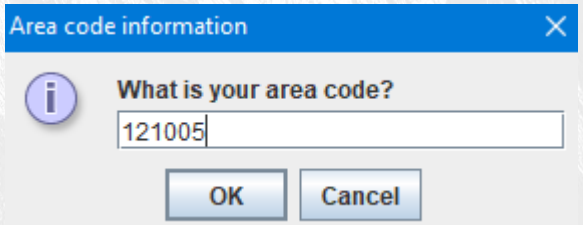
QUESTION_MESSAGE



WARNING_MESSAGE

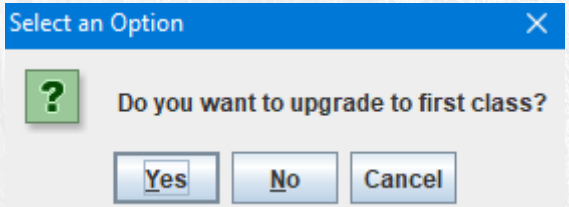


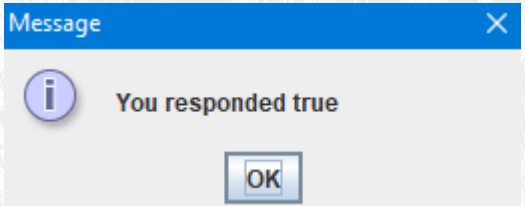
INFORMATION_MESSAGE



Confirmation Dialog

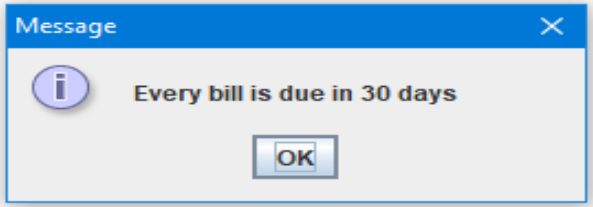
```
import javax.swing.JOptionPane;
//Program will return true or false on the base of the option which user had selected
public class ConfirmDialog {
    public static void main(String[] args)
    {
        int selection;
        boolean isYes;
        selection = JOptionPane.showConfirmDialog(null,"Do you want to upgrade to first class?");
        isYes = (selection == JOptionPane.YES_OPTION);
        JOptionPane.showMessageDialog(null,"You responded " + isYes);
    }
}
```





Concatenation

```
11  import javax.swing.JOptionPane;
12  public class Concatenation
13  {
14      public static void main(String[] args)
15      {
16          int billingDate = 5;
17          System.out.print("Bills are sent on day ");
18          System.out.print(billingDate);
19          System.out.println(" of the month");
20          System.out.println("Next bill: October " + billingDate);
21          int creditDays = 30;
22          JOptionPane.showMessageDialog(null, "Every bill is due in " + creditDays + " days");
23      }
24  }
25
```



Output - JavaExamples (run) #2

```
run:
Bills are sent on day 5 of the month
Next bill: October 5
```

Type Conversion

```
12    //Conversion of Double into integer ,long ,byte or vice-versa
13    public class TypeConversion {
14        public static void main(String[] args)
15        {
16            double d = 323.142;
17            //explicit type casting
18            long l = (long)d;
19            int i = (int)d;
20            byte b = (byte)d;
21            System.out.println("Double value = "+d);
22            System.out.println("Long value = "+l);
23            System.out.println("Int value = "+i);
24            System.out.println("Byte value = "+b);
25    //          example2();
26        }
27        static void example2(){
```

Output - JavaExamples (run)

```
run:
Double value = 323.142
Long value = 323
Int value = 323
Byte value = 67
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method Calling

```
12  public class MethodCalling {
13      public static void main(String[] args)
14      {
15          myStaticMethod(); // Call the static method
16  //          myPublicMethod(); This would compile an error
17          MethodCalling myObj = new MethodCalling(); // Create an object of Main
18          myObj.myPublicMethod(); // Call the public method on the object
19          System.out.println("Accessing Objects From Within Same Class => ");
20          myObj.speed(200);
21          SecondClass secondclass = new SecondClass(); // Create a SecondClass Java Class object
22          System.out.println("Accessing Objects From Second Different Class => ");
23          secondclass.fullThrottle();       // Call the fullThrottle() method
24          secondclass.speed(200);           // Call the speed() method
25      }
26  //    Static method
27      static void myStaticMethod()
28      {
29          System.out.println("Static methods can be called without creating objects");
30      }
31  //    Public method
32      public void myPublicMethod()
33      {
34          System.out.println("Public methods must be called by creating objects");
35      }
36  //    Create a speed() method and add a parameter
37      public void speed(int maxSpeed) {
38          System.out.println("Max speed is: " + maxSpeed);
39      }
40  }
```

Method Calling - Continue

```
12  public class SecondClass {
13      public void fullThrottle() {
14          System.out.println("The car is going as fast as it can!");
15      }
16      public void speed(int maxSpeed) {
17          System.out.println("Max speed is: " + maxSpeed);
18      }
19  }
20
```

Output - JavaExamples (run)

```
run:
Static methods can be called without creating objects
Public methods must be called by creating objects
Accessing Objects From Within Same Class =>
Max speed is: 200
Accessing Objects From Second Different Class =>
The car is going as fast as it can!
Max speed is: 200
BUILD SUCCESSFUL (total time: 0 seconds)
```

Constructor

```
12    //Constructors are used to initialize the instances of your classes.
13    //You use a constructor to create new objects often with parameters specifying the initial state
14    //or other important information about the object
15    //Parameterized Constructor.
16    public class ConstructorExample {
17    //     data members of the class.
18        int modelYear;
19        String modelName;
20        /* constructor would initialize data members with the values of passed arguments
21        while object of that class created. */
22    //     Constructor with two arguments
23        public ConstructorExample(int year, String name) {
24            modelYear = year;
25            modelName = name;
26        }
27    //     Constructor with one argument
28        public ConstructorExample(int year) {
29            modelYear = year;
30        }
31        public static void main(String[] args) {
32    //          this would invoke the parameterized constructor.
33            ConstructorExample myCar = new ConstructorExample(1969, "Mustang");
34            System.out.println("ConstructorExample with two arguments = " + myCar.modelYear + " " + myCar.modelName);
35            ConstructorExample myCar1 = new ConstructorExample(1981);
36            System.out.println("ConstructorExample with one argument = " + myCar1.modelYear);
37        }
38    }
```

Output - JavaExamples (run)

```
run:
ConstructorExample with two arguments = 1969 Mustang
ConstructorExample with one argument = 1981
BUILD SUCCESSFUL (total time: 0 seconds)
```

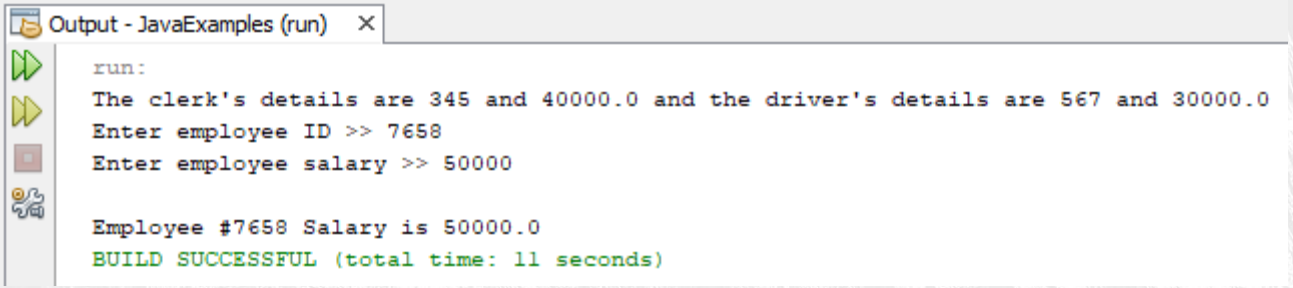
Getter Setter – Model Class

```
12  public class Employee {
13      private int empNum;
14      private String empLastName;
15      private String empFirstName;
16      private double empSalary;
17      public int getEmpNum()
18      {
19          return empNum;
20      }
21      public void setEmpNum(int emp)
22      {
23          empNum = emp;
24      }
25      public String getEmpLastName()
26      {
27          return empLastName;
28      }
29      public void setEmpLastName(String name)
30      {
31          empLastName = name;
32      }
33      public String getEmpFirstName()
34      {
35          return empFirstName;
36      }
37      public void setEmpFirstName(String name)
38      {
39          empFirstName = name;
40      }
41      public double getEmpSalary()
42      {
43          return empSalary;
44      }
45      public void setEmpSalary(double sal)
46      {
47          empSalary = sal;
48      }
49  }
```

Getter Setter – Main Class

```
14  public class GetterSetter {
15      public static void main(String[] args){
16          Employee clerk = new Employee();
17          Employee driver = new Employee();
18          clerk.setEmpNum(345);
19          clerk.setEmpSalary(40000);
20          driver.setEmpNum(567);
21          driver.setEmpSalary(30000);
22          System.out.println("The clerk's details are " + clerk.getEmpNum() + " and " + clerk.getEmpSalary() + " and the drive↩
    r's details are " + driver.getEmpNum()+ " and " + driver.getEmpSalary());
23  //          Employee is declared.
24          Employee myEmployee;
25  //          Value returned from getEmployeeData() method is assigned to myEmployee object.
26          myEmployee = getEmployeeData();
27  //          myEmployee object is passed to displayEmployee() method.
28          displayEmployee(myEmployee);
29      }
30      public static Employee getEmployeeData(){
31  //          Return type is Employee.
32          Employee tempEmp = new Employee();
33          int id;
34          double sal;
35          Scanner input = new Scanner(System.in);
36          System.out.print("Enter employee ID >> ");
37          id = input.nextInt();
38          tempEmp.setEmpNum(id);
39          System.out.print("Enter employee salary >> ");
40          sal = input.nextDouble();
41          tempEmp.setEmpSalary(sal);
42          return tempEmp;
43      }
44      public static void displayEmployee(Employee anEmp)
45      {
46          System.out.println("\nEmployee #" + anEmp.getEmpNum() + " Salary is " + anEmp.getEmpSalary());
47      }
48  }
```

Getter Setter – Output



THANK YOU!!