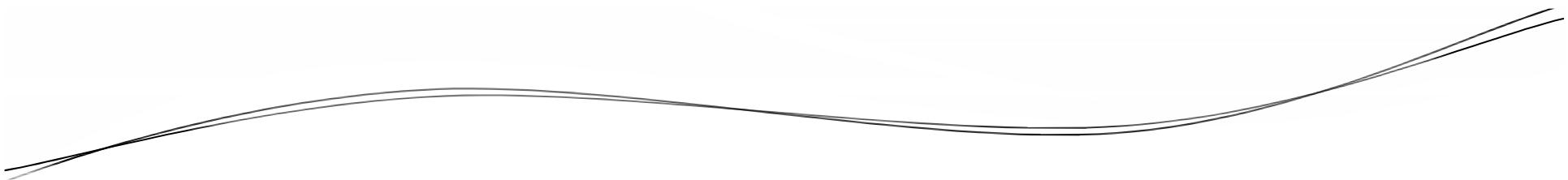


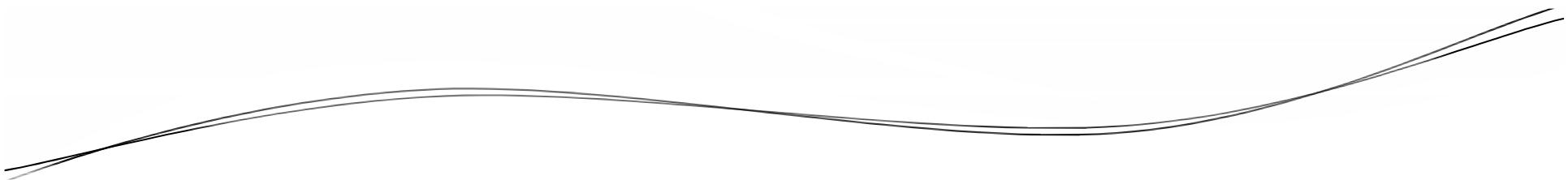
Introduction to Java

By Rahul Barve



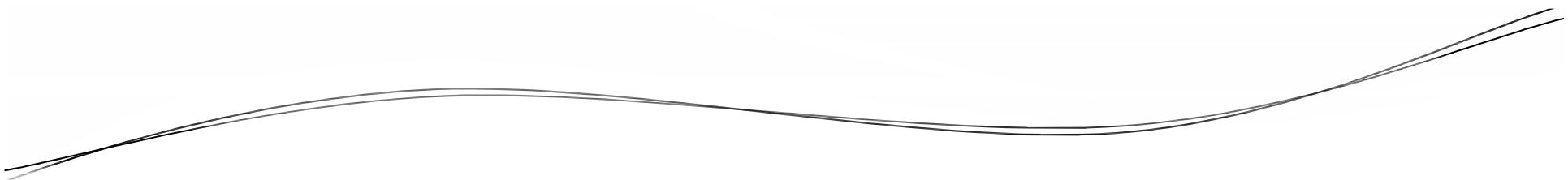
Objectives

- Introduction to Java
- Object Oriented Concepts
- A First Java Program
- Understanding JRE
- Data Types in Java



Introduction to Java

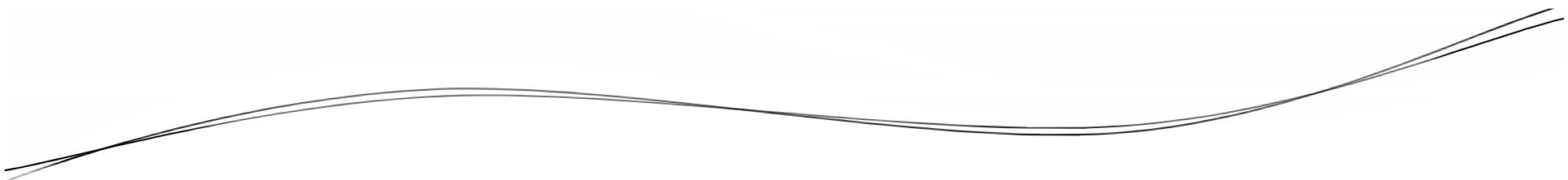
By Rahul Barve



Introduction to Java

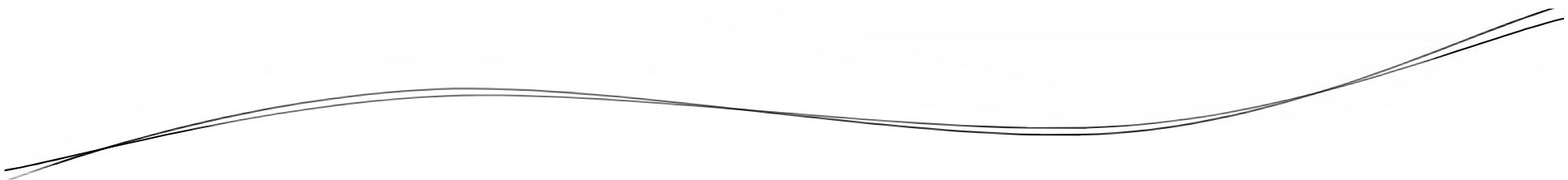
- Java is an object oriented programming language designed to build real time business applications.
- It is a language that follows a notion:

Write Once Run Anywhere



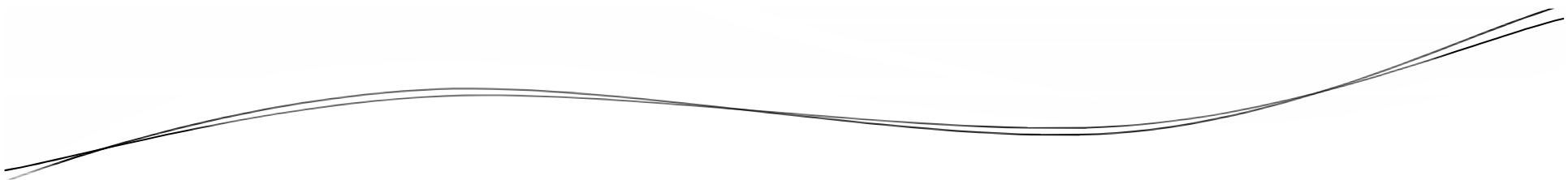
Introduction to Java

- Suitable for building applications like:
 - Standalone or Desktop
 - Network Based
 - Web Based (Internet or Intranet)
 - Mobile Based (Android)



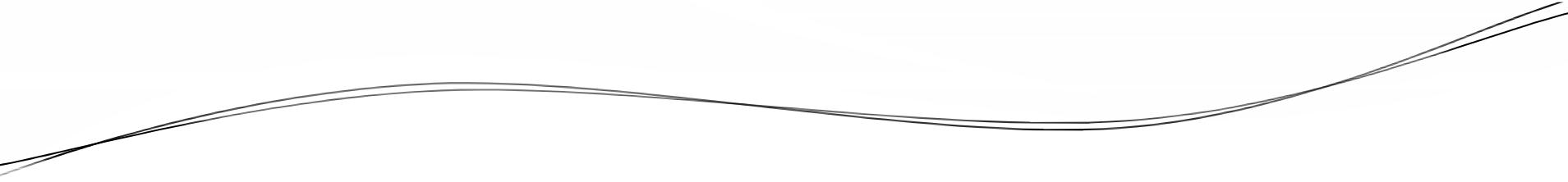
Brief History

By Rahul Barve



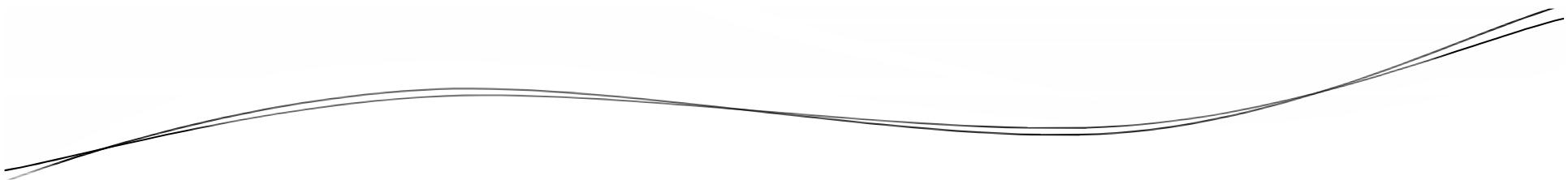
Brief History

- Java was invented by James Gosling in coordination with Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems in 1991.
- Initially was called OAK but renamed to JAVA in 1995.



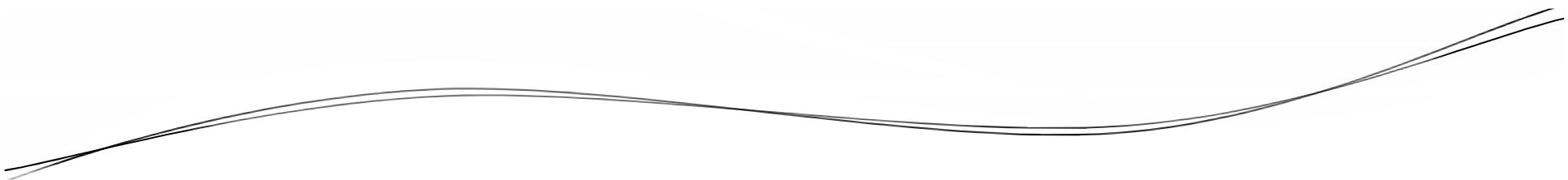
Java Buzzwords

By Rahul Barve



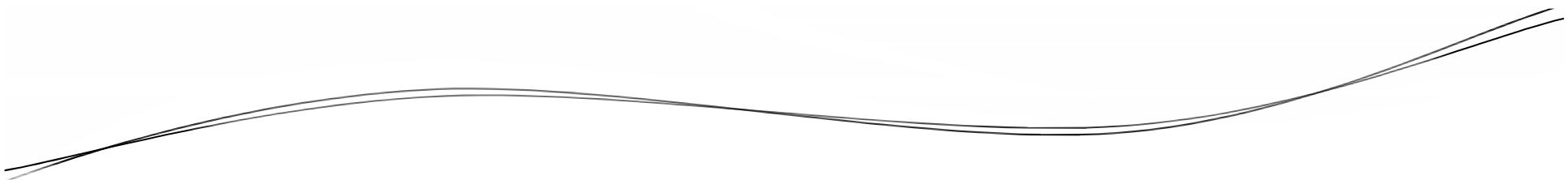
Java Buzzwords

- Simple
- Object Oriented
- Portable
- Robust
- Secure
- Multithreaded
- Architecture Neutral
- Interpreted
- Distributed



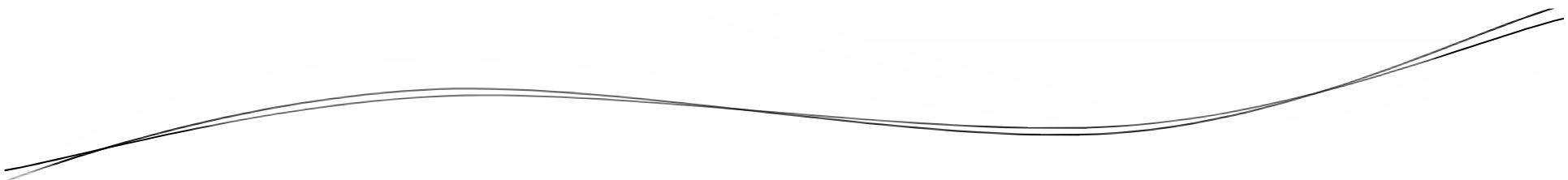
Object Oriented Programming

By Rahul Barve



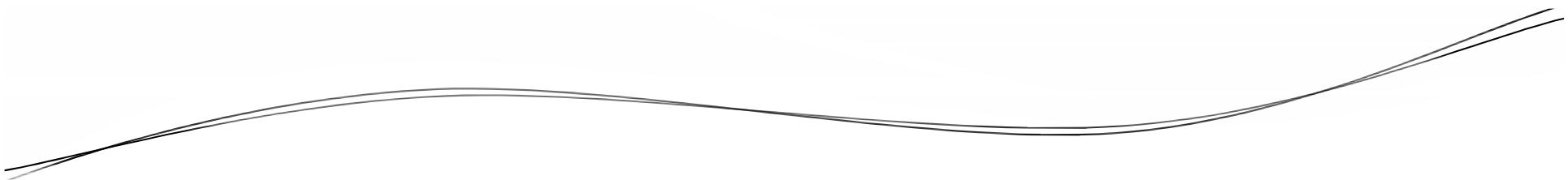
Object Oriented Programming

- Object Oriented Programming is a set of principles used to design the software with the help of business domain specific entities known as Objects.
- E.g. Account in a Banking system, Employee in a HR system, Patient in a Hospital system, Book in a Library system and so on.



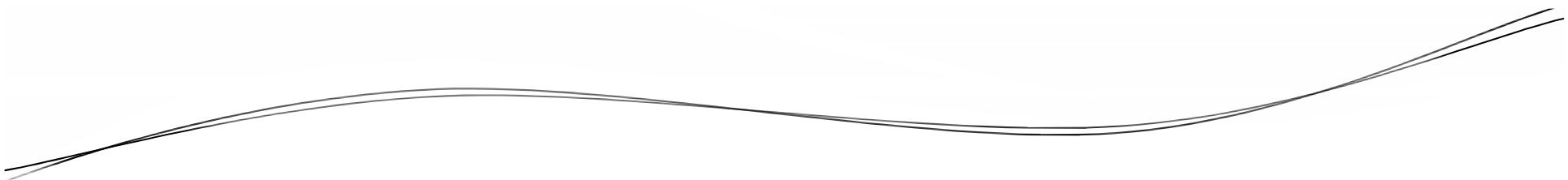
Object Oriented Programming

- An Object is an entity having a well defined structure.
- Typically an object has a state and behaviors.



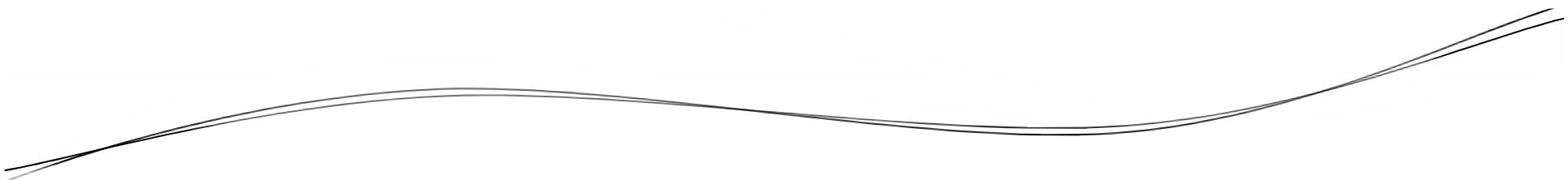
OOP Principles

By Rahul Barve



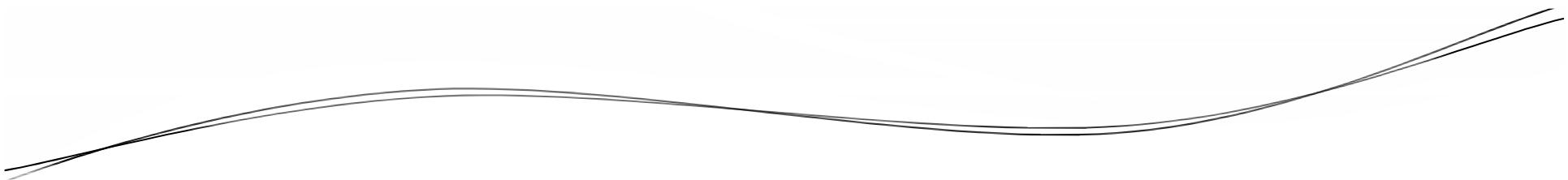
OOP Principles

- Abstraction
- Encapsulation
- Modularity
- Inheritance
- Polymorphism



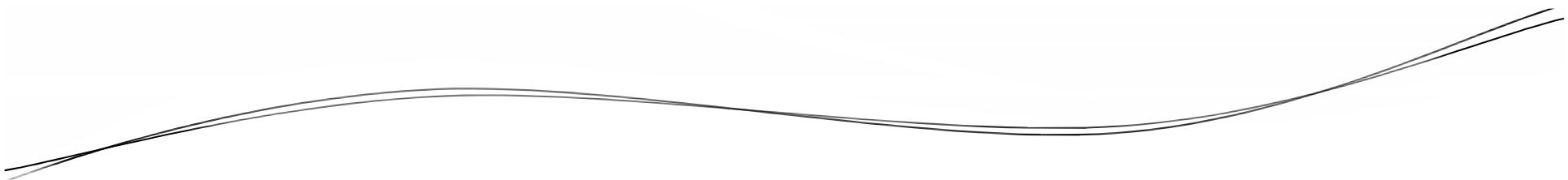
Abstraction

- The process of identifying key aspects and ignoring rest is known as abstraction.
- Only domain expertise can do right abstraction.



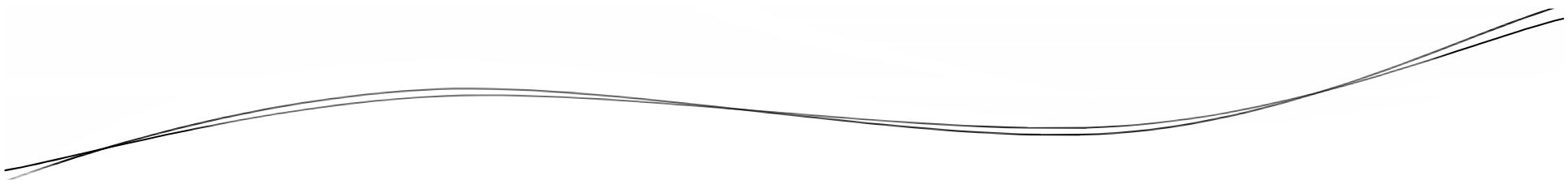
Encapsulation

- It provides a separation between an abstraction and its implementation.
- Ensures that the data manipulation does not take place directly.



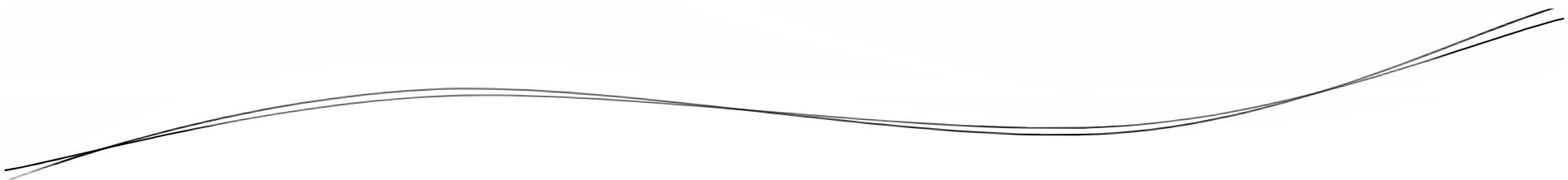
Modularity

- It is the process of breaking up the system into small units of work, referred as modules.
- Promotes loose coupling.
- Brings flexibility and Reusability.



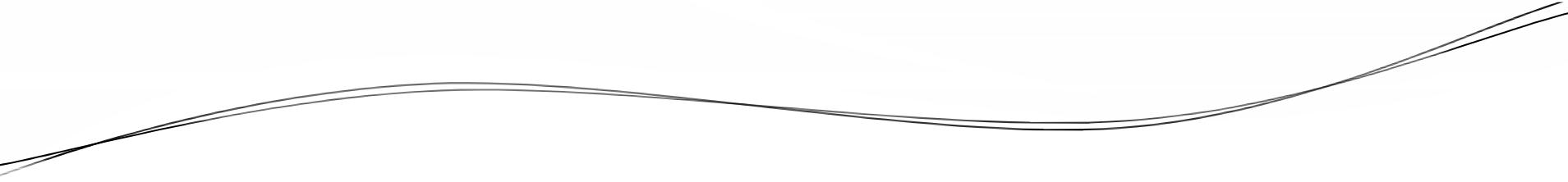
Inheritance

- It is the process of building a new structure based upon the existing one.
- Already built structure can be extended in any direction as and when required.
- It represents IS-A relationship.



Polymorphism

- It refers to many forms.
- Objects responding to same message in different ways is known as polymorphism.

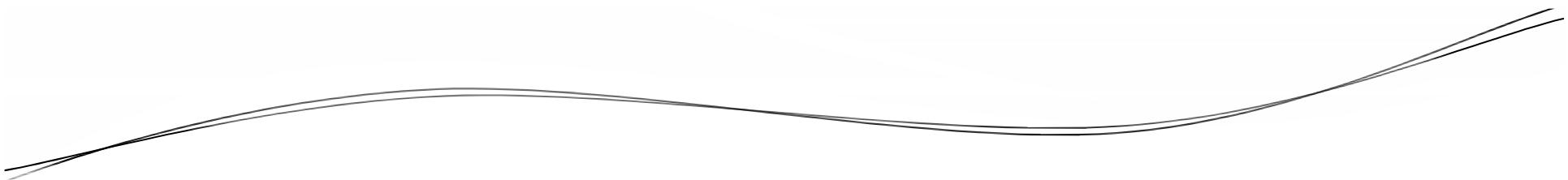


A First Java Program

By Rahul Barve

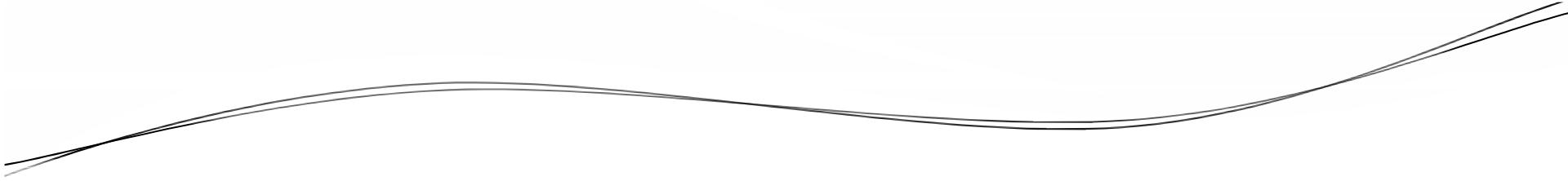
A First Java Program

```
//HelloWorld.java
public class HelloWorld {
    public static void main(String args[] ) {
        System.out.println("Hello World");
    }
}
```



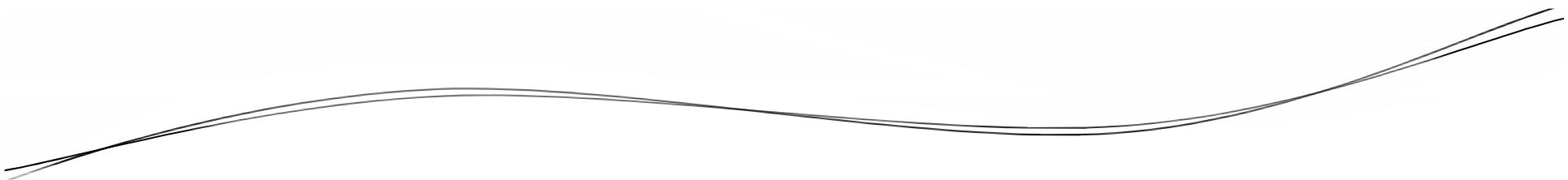
Basic Rules and Concepts

By Rahul Barve



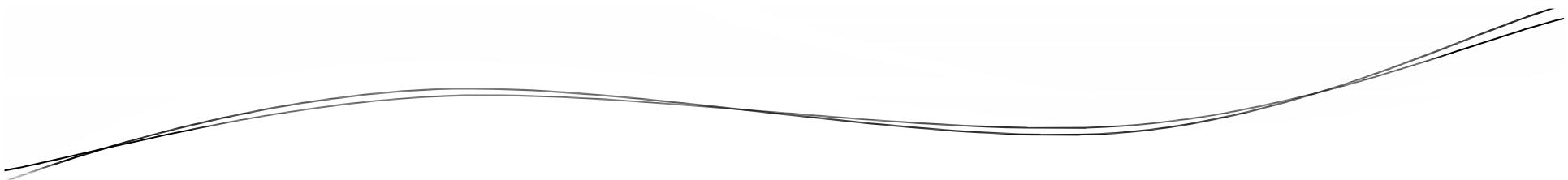
Basic Rules and Concepts

- All Java source files have .java extension.
- If a class is declared as public, the source file name must match the name of the class.



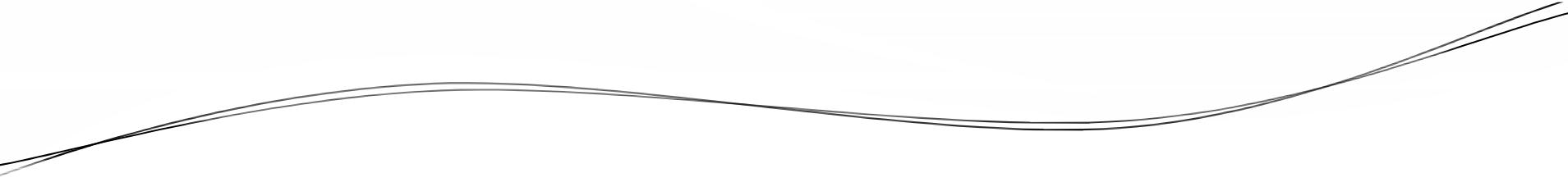
Basic Rules and Concepts

- A source file may have multiple classes defined provided, not more than one classes are declared as public.
- When a source file is compiled, a .class file is generated for every class.



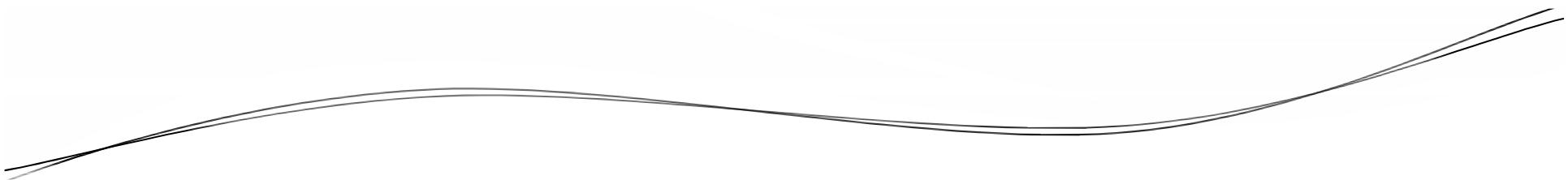
Java's Magic: The Bytecode

- When a source file is compiled, a .class file is created which contains a bytecode.
- A bytecode is a highly optimized set of instructions designed to be executed by the Java runtime system.



Java Runtime Environment

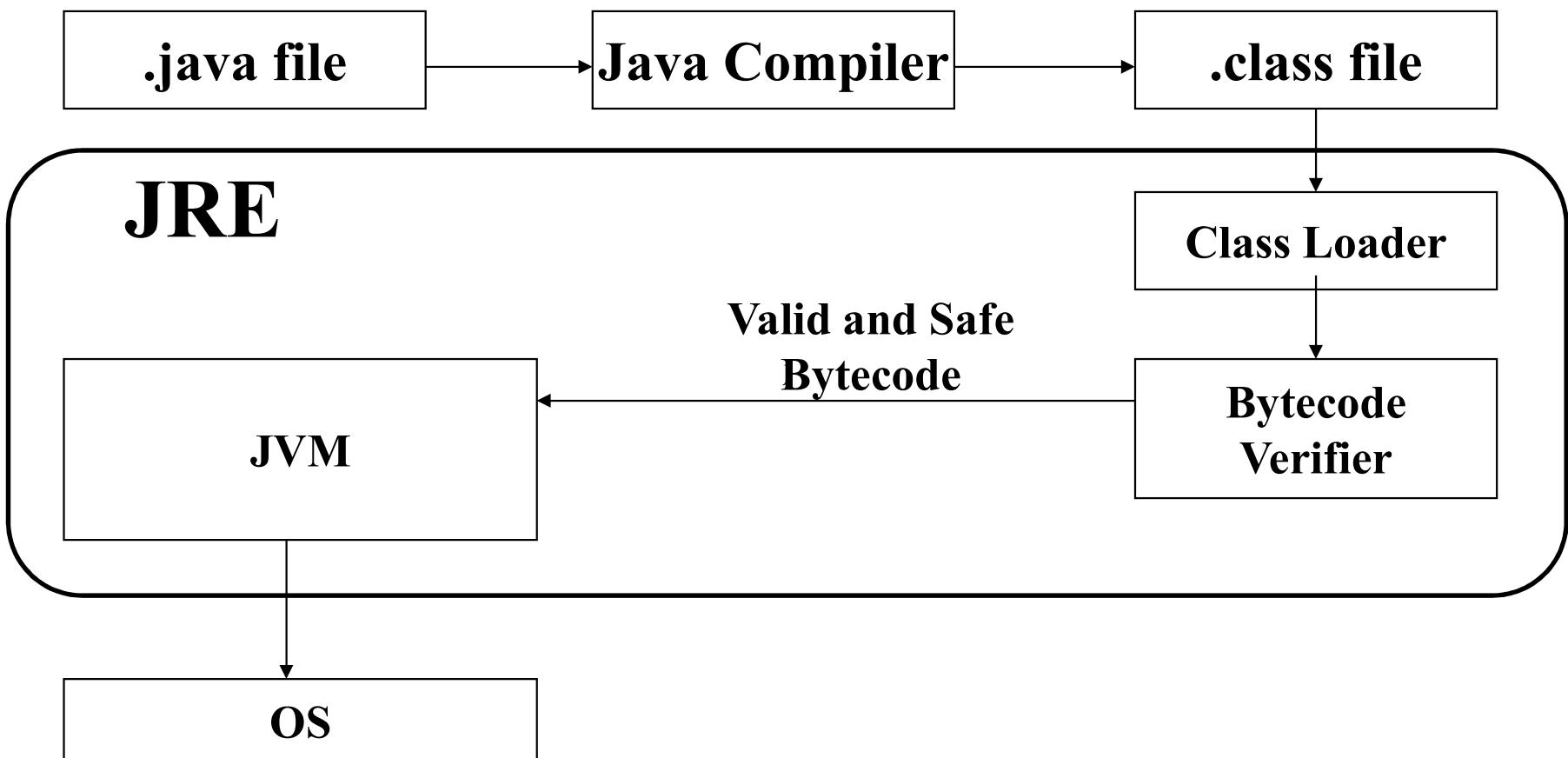
By Rahul Barve

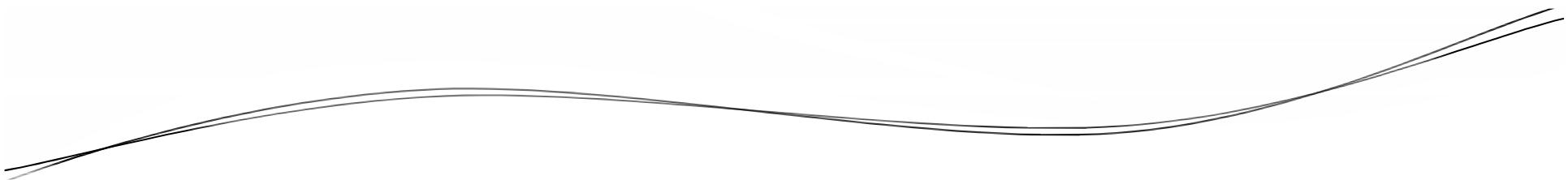


Java Runtime Environment

- It is a basic environment required to execute a Java program.
- JRE mainly consists of:
 - Class Loader
 - Bytecode Verifier
 - Java Virtual Machine

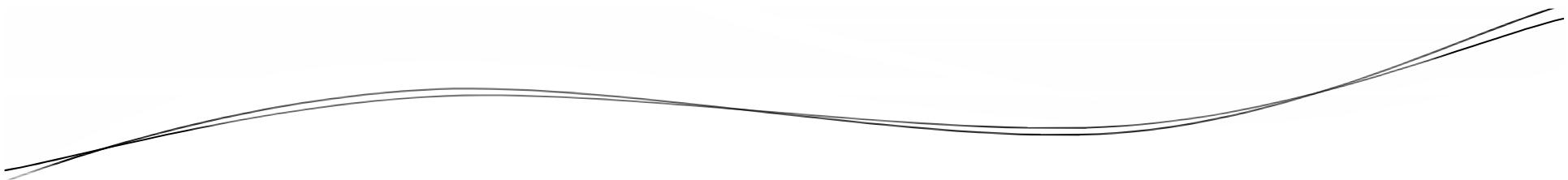
Java Runtime Environment





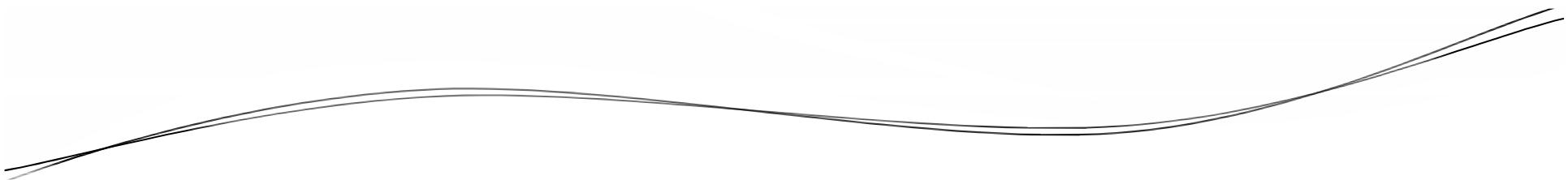
Java Primitive Types

By Rahul Barve



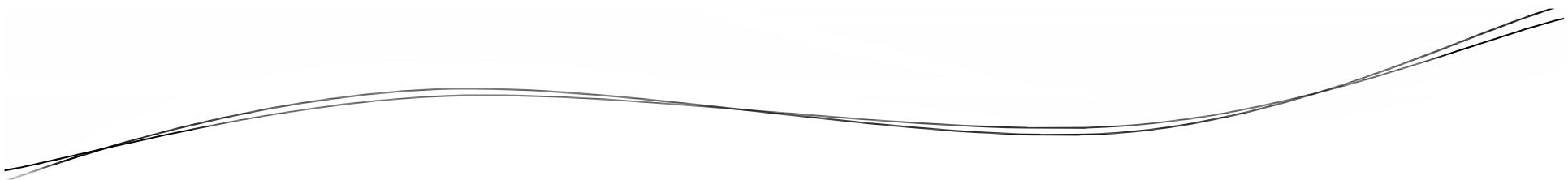
Java Primitive Types

- Java is a strongly typed language.
- Every variable, expression has a type.
- All assignments, whether explicit or via parameter passing, are checked for type compatibility.



Java Primitive Types

- Integers
 - byte (1), short (2), int (4), long (8)
- Decimals
 - float (4), double (8)
- Characters
 - char (2)
- Booleans
 - boolean (1 bit)



Lets Summarize

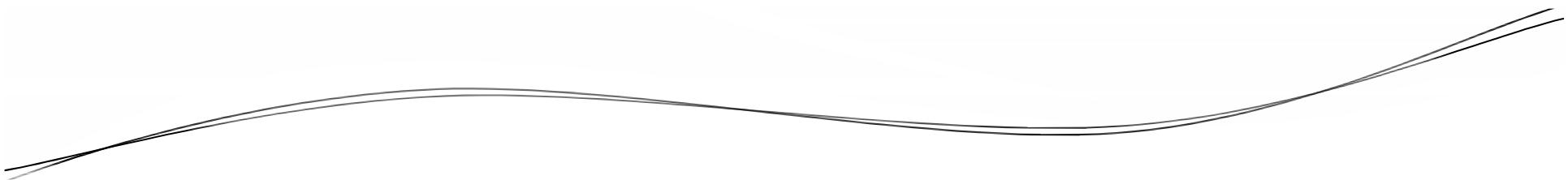
- Introduction to Java
- Object Oriented Concepts
- A First Java Program
- Understanding JRE
- Data Types in Java

PrimitiveTypeExample.java SimpleArrayExample.java

```
1 public class PrimitiveTypeExample {
2
3     public static void main(String[] args) {
4         System.out.println("Demonstrating Primitive Types: ");
5         int val = 100;
6         boolean success = true;
7         float price = 65.25f;
8         char ch = 'A';
9         String name = "Joker"; //String is a pre-defined class
10
11        System.out.println("Value of val is " + val);
12        if(success)
13            System.out.println("I got success");
14        else
15            System.out.println("I got failure");
16
17        System.out.println("Welcome " + name);
18
19        System.out.println("First alphabet: " + ch);
20
21        System.out.println("Price for 10 items: " + (price * 10));
22
23    }
24
25
26
27 }
```

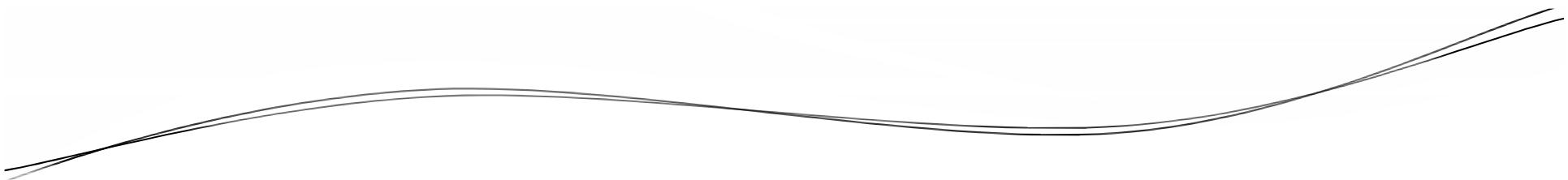
PrimitiveTypeExample.java SimpleArrayExample.java

```
1
2 public class SimpleArrayExample {
3
4     public static void main(String[] args) {
5         // Declaring an array of 3 integers
6         int numbers[] = new int[3];
7         //Initializing the array with some values
8         numbers[0] = 75;
9         numbers[1] = 275;
10        numbers[2] = 175;
11        for(int index=0;index<3;index++) {
12            int num = numbers[index];
13            System.out.println(num);
14        }
15
16        //Declaring and Initializing an array of Strings simultaneously
17        String riverNames[] =
18            {"Ganga", "Yamuna", "Kaveri", "Godavari", "Brahmaputra"};
19        //Obtaining the size of the array
20        int size = riverNames.length;
21        for(int index = 0; index < size; index++)
22            System.out.println(riverNames[index]);
23        System.out.println("-----");
24        //Printing the river names using Enhanced For Loop (For-Each)
25        for(String river : riverNames)
26            System.out.println(river.toUpperCase());
27
28    }
29
30 }
```



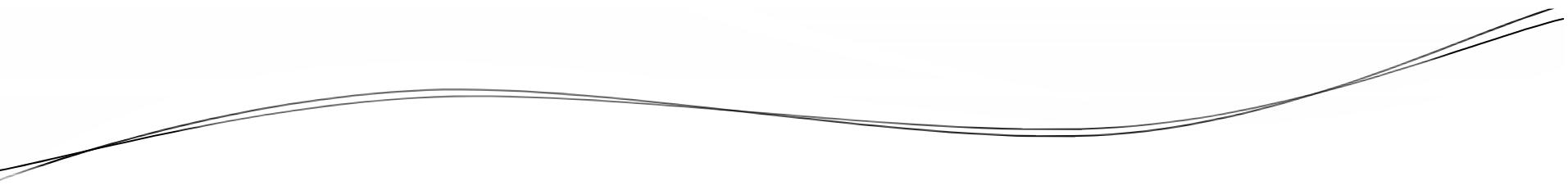
Classes in Java

By Rahul Barve



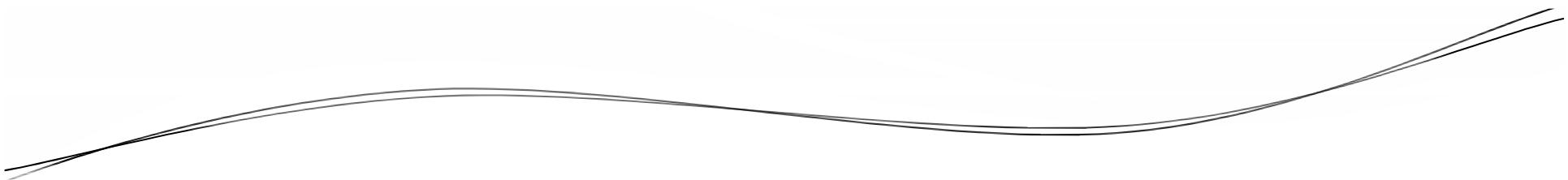
Objectives

- Class Basics
- Object Creation
- Access Modifiers
- Working with Methods
- Method Overloading
- Working with Constructors
- Understanding this
- Understanding Static Members
- Variable Types



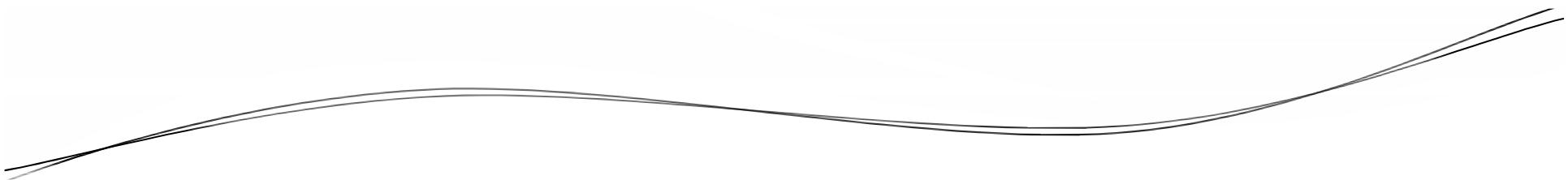
What is Class

By Rahul Barve



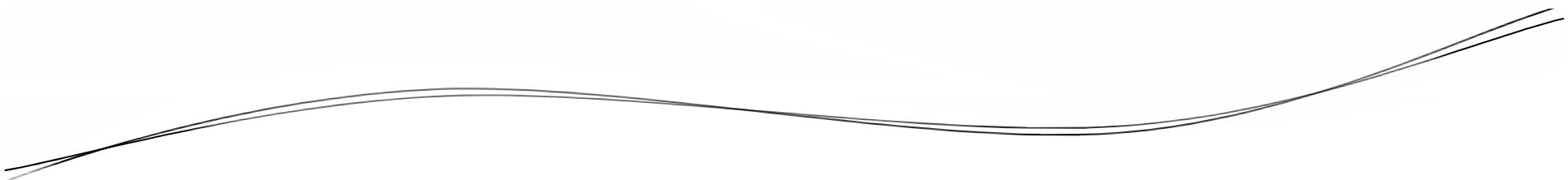
What is Class

- A class is at the core of Java.
- Any concept to be implemented, must be encapsulated within a class.



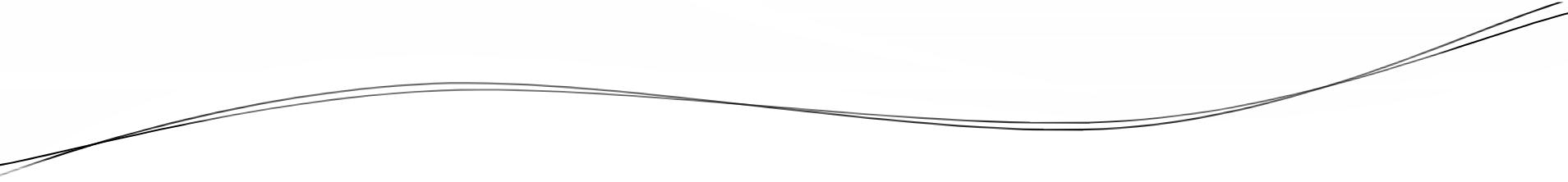
What is Class

- A Class is a template that decides a structure for an object.
- Using classes and objects, one can map 2 major pillars of OOP: Abstraction and Encapsulation.



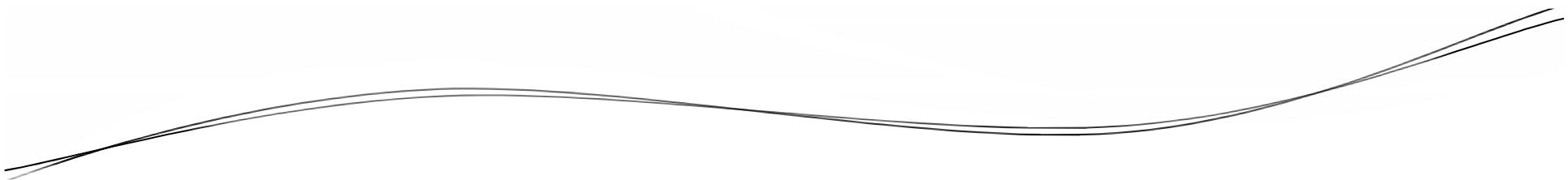
Class Syntax

```
class <class-name> {  
    <variable-declarations>  
    <method-definitions>  
}
```



A Simple Class

```
class Planet{  
    String name;  
    int moons;  
}
```



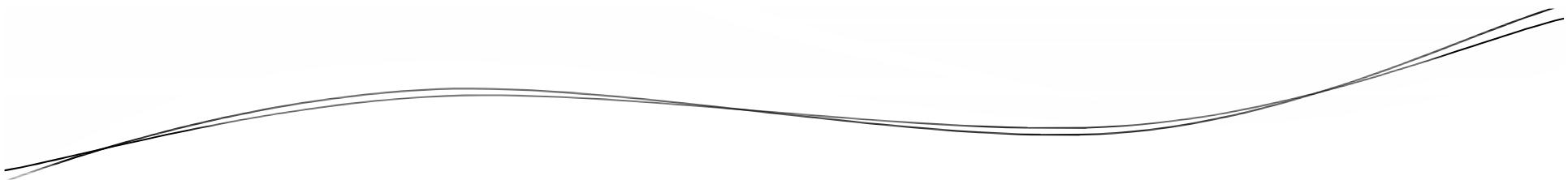
Creating Object

By Rahul Barve

Creating Object

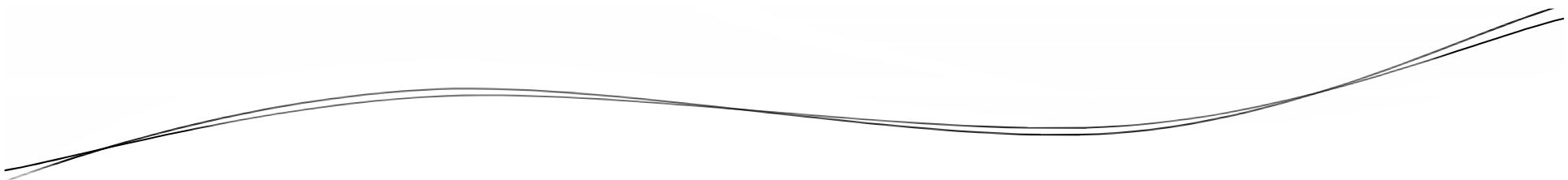
- Once a class is created, it can be further used by creating its object.
- Syntax:

```
<class-name> <ref-var-name> =  
    new <class-name>() ;
```



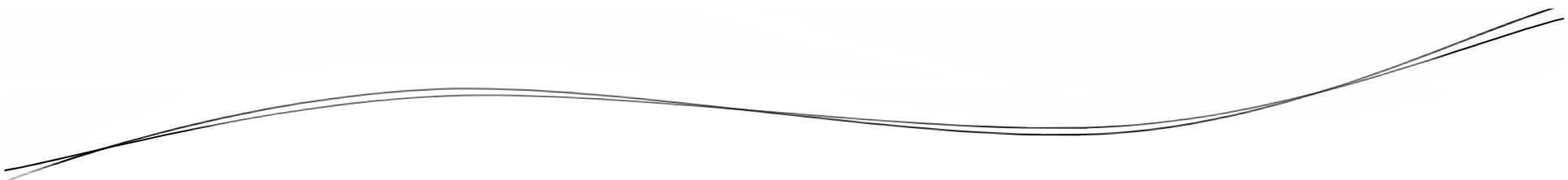
Access Modifiers

By Rahul Barve



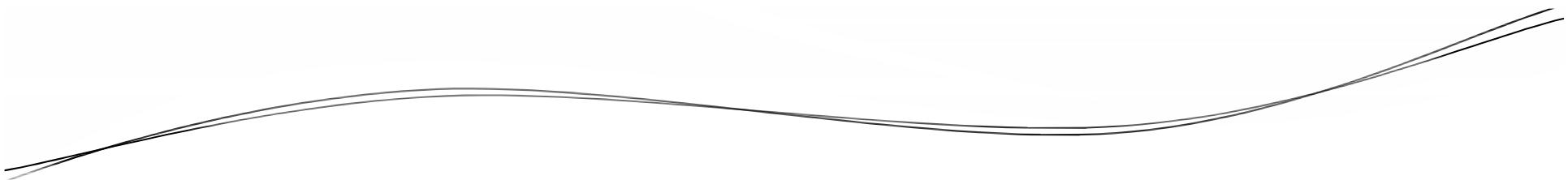
Access Modifiers

- Access modifiers are used to specify the scope of class members.
- These are private, public, protected and default.



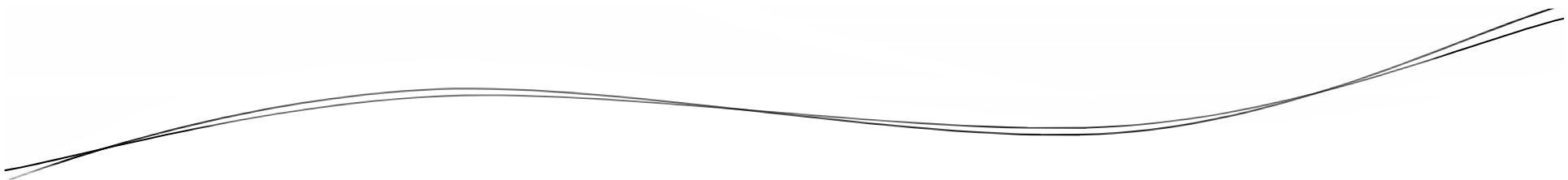
Adding Methods

By Rahul Barve



Adding Methods

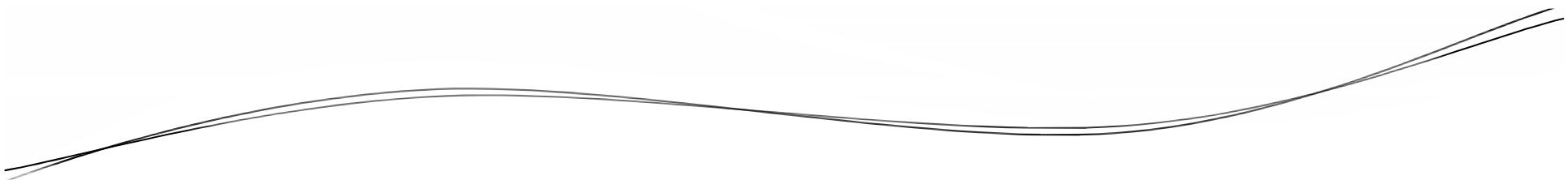
- Once an object is created, at any time it can be manipulated with the help of methods.
- Methods act like behaviors or operations.



Adding Methods

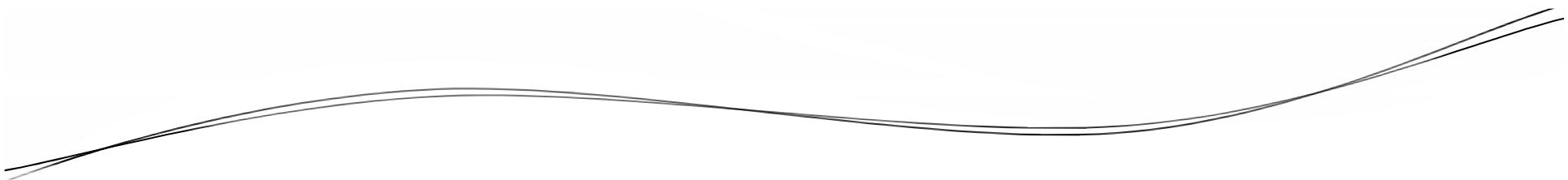
- Syntax:

```
<return-type> <method-name> ( [param-list] ) {  
    //Some Code  
}
```



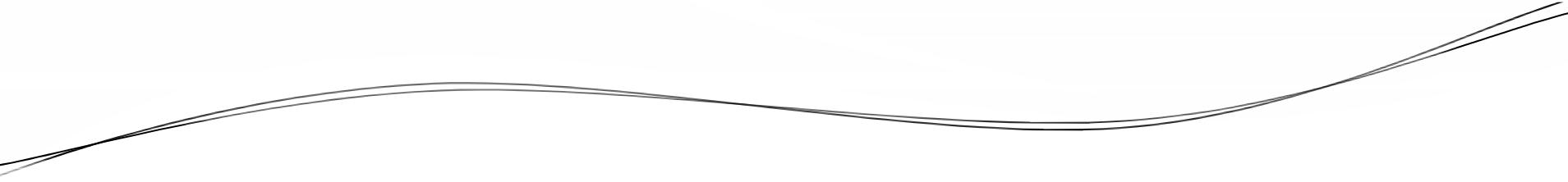
Accessor and Mutator Methods

By Rahul Barve



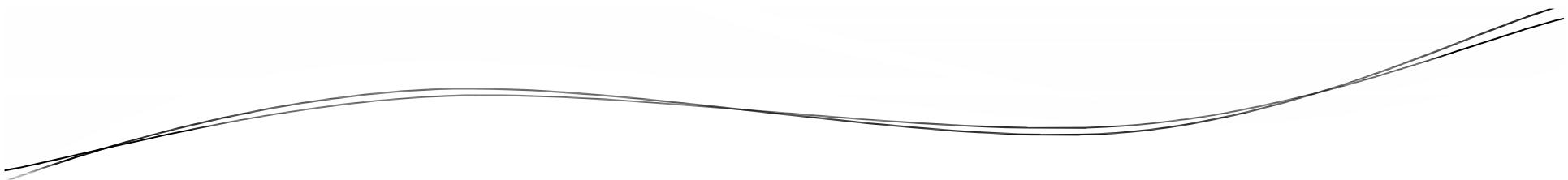
Accessor and Mutator Methods

- Used to retrieve or modify the values of the fields for a specific object.
- These methods follow the convention:
`getXXX()` and `setXXX()`



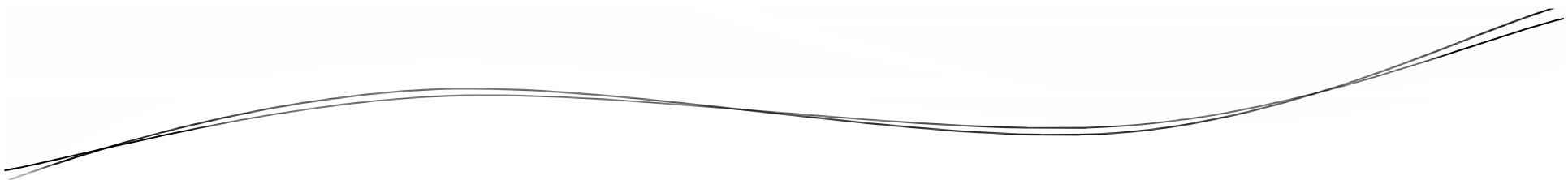
Method Overloading

By Rahul Barve



Method Overloading

- If 2 or multiple methods have same name but different signatures, then those methods are called as overloaded methods.
- It's a compile time polymorphism.



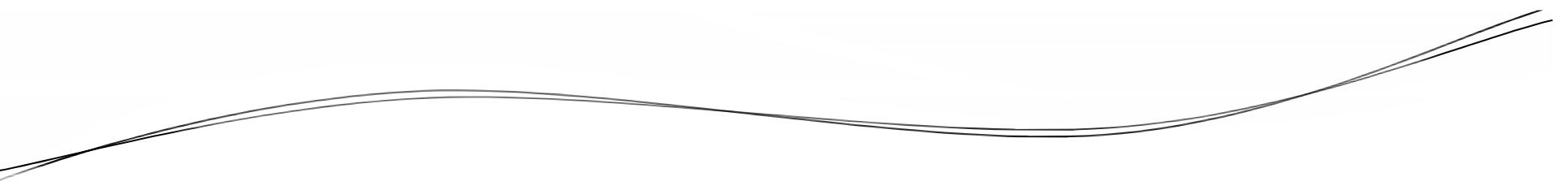
Method Overloading

- E.g.

```
class Test {  
    void test() { }  
    void test(int a) { }  
    void test(int a, int b) { }  
    void test(int a, String b) { }  
    void test(String a, int b) { }  
}
```

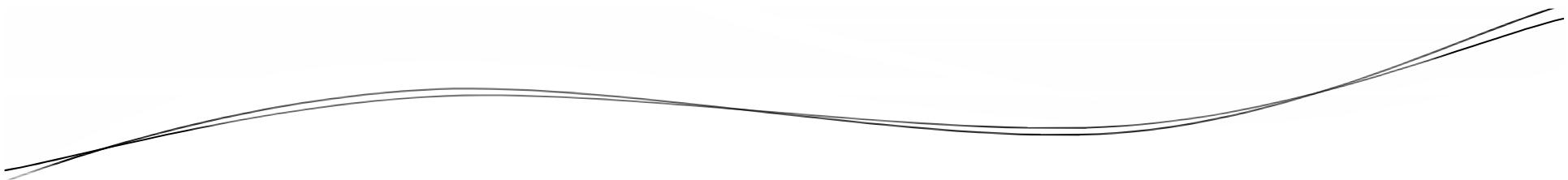
Method Overloading

```
Test t = new Test();  
t.test();  
t.test(10);  
t.test(10,20);  
t.test(10, "Hello");  
t.test("Welcome", 20);  
t.test("Hello", "Welcome"); → Error
```



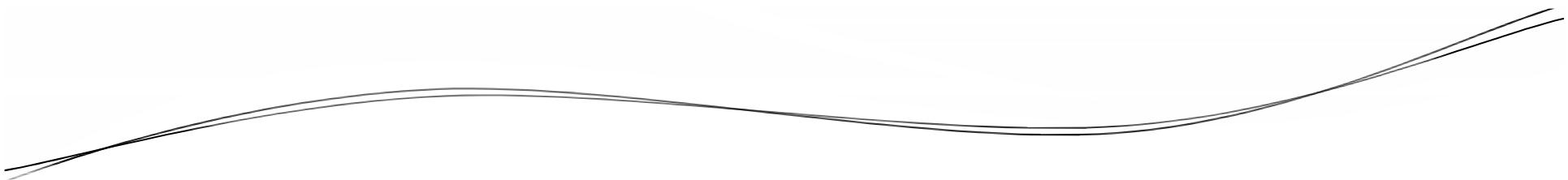
Constructor

By Rahul Barve



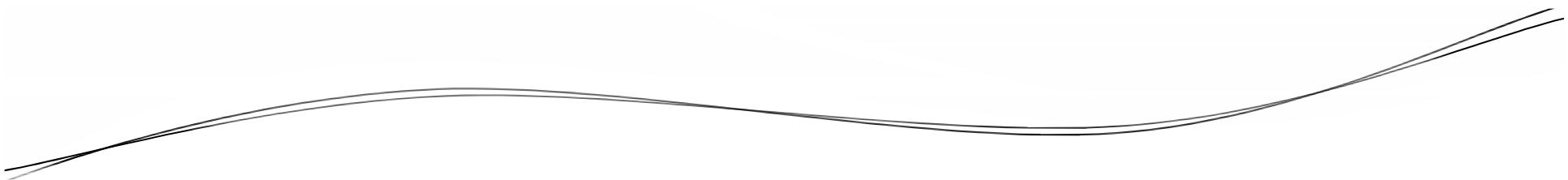
Constructor

- Constructor is a special member within a class having same name as that of a class name.
- It is invoked implicitly as soon as an object is created.



Constructor

- Does not have any return type.
- Constructors are used for object initialization.



Constructor

- A constructor without any parameter is known as no-argument constructor.
- Constructors can be overloaded.

Constructor

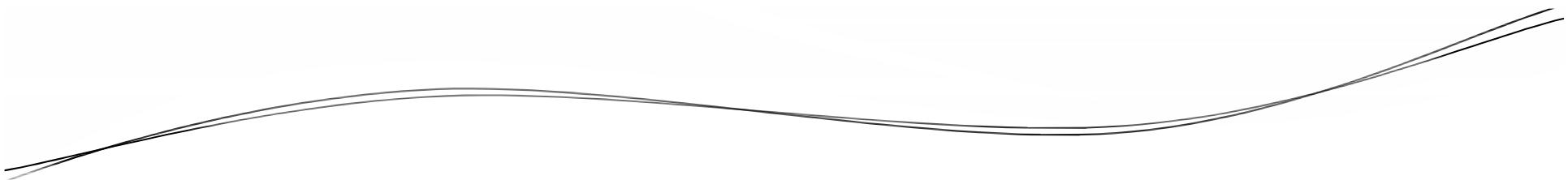
```
class Box {  
    int length, width, height;  
    Box() { //No-Argument  
        length = 10;  
        width = 8;  
        height = 5;  
    }  
    Box(int l, int w, int h) { //Parameterized  
        length = l;  
        width = w;  
        height = h;  
    }  
}
```

Constructor

```
Box box1 = new Box();  
//Invokes no-argument constructor
```

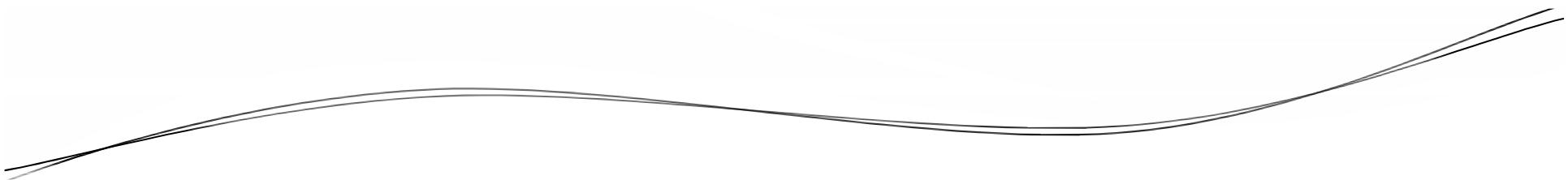
```
Box box2 = new Box(20,15,12);  
//Invokes parameterized constructor
```

```
Box box3 = new Box(12);  
//Error
```



this reference

By Rahul Barve

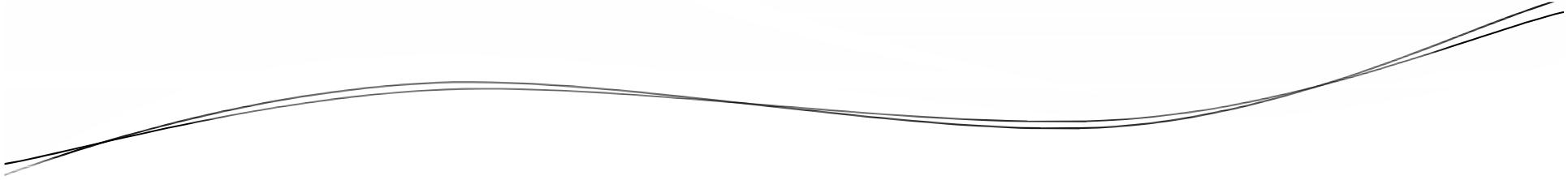


'this' reference

- Every class member function gets a hidden parameter known as `this` reference.
- `this` is a keyword in Java.
- It always refers to the object that is currently invoked.

'this' reference

```
class Box {  
    int length, width, height;  
    Box() {  
        length = 10;  
        //Is equivalent to  
        //this.length = 10;  
        ... . . .  
    }  
}
```



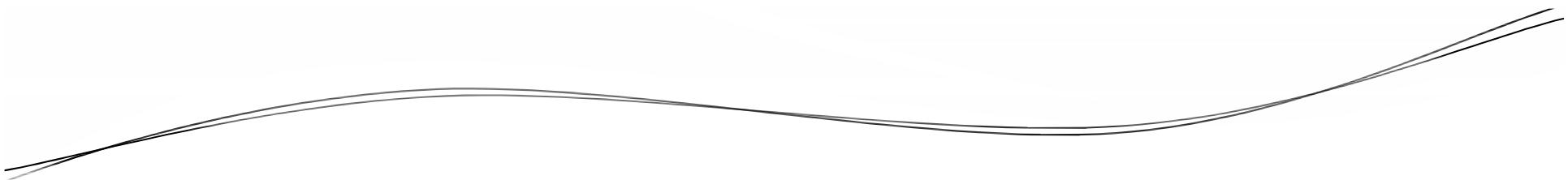
'this' reference

- It becomes mandatory to use this reference when local variable names conflict with instance variable names.
- It is used to resolve the scope.

'this' reference

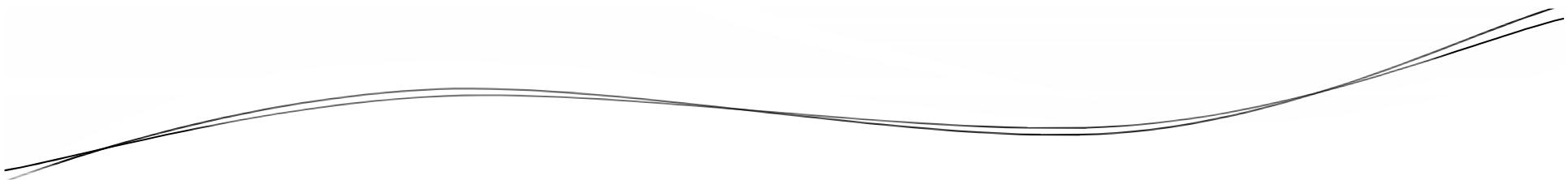
```
class Person {  
    String name;      //Instance Variable  
    Person(String name) { //Local Variable  
        //name = name; //Not OK  
        this.name = name; //OK  
    }  
}
```

The diagram illustrates the scope of variables in Java. It shows a class named 'Person' with an instance variable 'name'. Inside the constructor, there is a local variable 'name' which shadows the instance variable. The code attempts to assign the local variable to itself, which is marked as 'Not OK'. Instead, the correct assignment 'this.name = name;' is shown, which is marked as 'OK'.



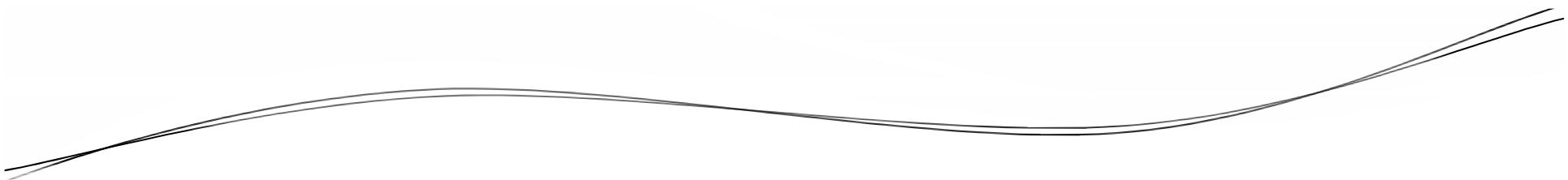
Static Members

By Rahul Barve



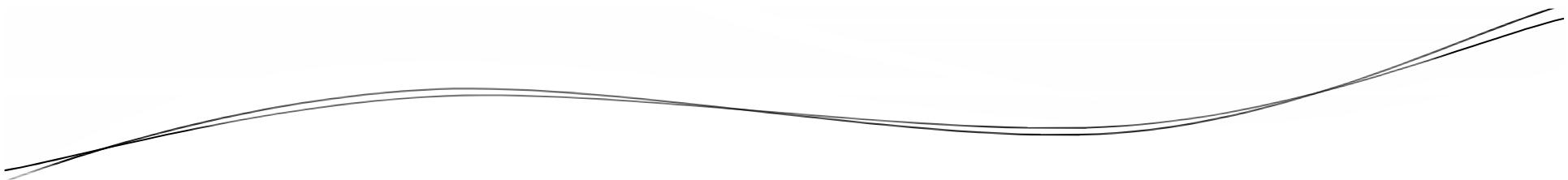
Static Members

- Whatever declarations, definitions are done within a class are collectively called as members of a class e.g. variables, methods and constructors.



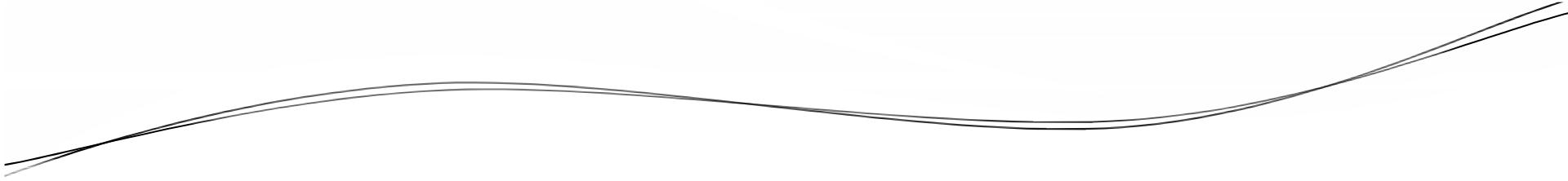
Static Members

- Members of a class are of 2 types:
 - Non-Static – These are associated with a particular instance of a class.
 - Static – These are not associated with any particular instance; rather they are associated with the whole class.



Static Members

- Static members are either variables or methods.
- Constructors cannot be declared static.



Static Members

- A static variable has a single copy irrespective of the number of objects created.
- Hence they are also known as class variables.

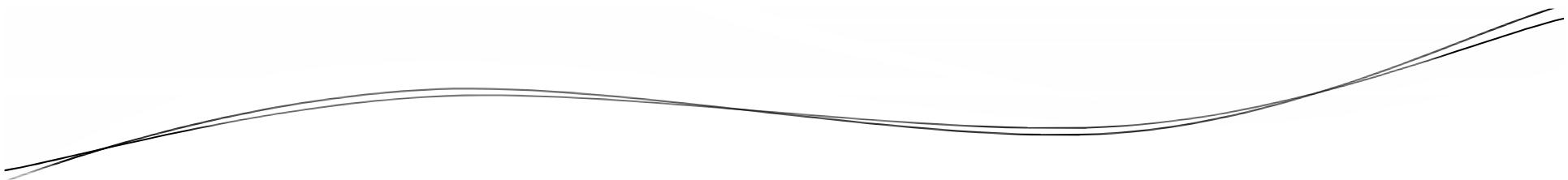
Static Members

```
class Person {  
    //Instance Variables  
    String name;  
    int age;  
  
    //Class Variables  
    static float avgAge;  
    static int personCount;  
    ...  
}
```

Static Members

- A static modifier is also used to introduce global variables.
- E.g.

```
public class Math {  
    public static float PI = 3.14f;  
    //Can be accessed from anywhere  
    //by using Math.PI  
}
```

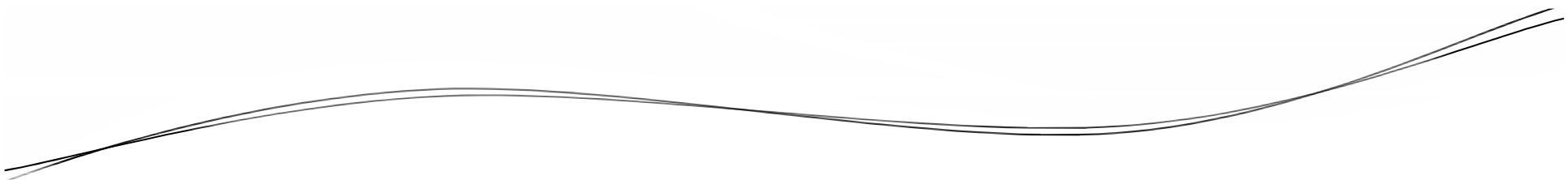


Static Members

- Like variables, methods can also be declared as static.
- Can be invoked without any object.

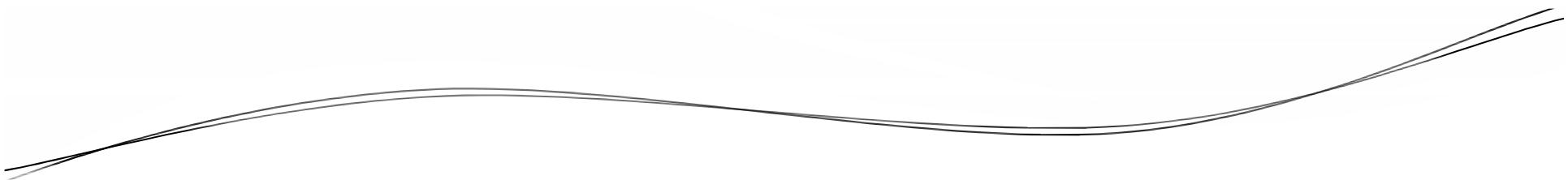
Static Members

```
class Math {  
    static int getFactorial(int num) {  
        //Code to get factorial  
    }  
}  
  
//int fact = Math.getFactorial(5);
```



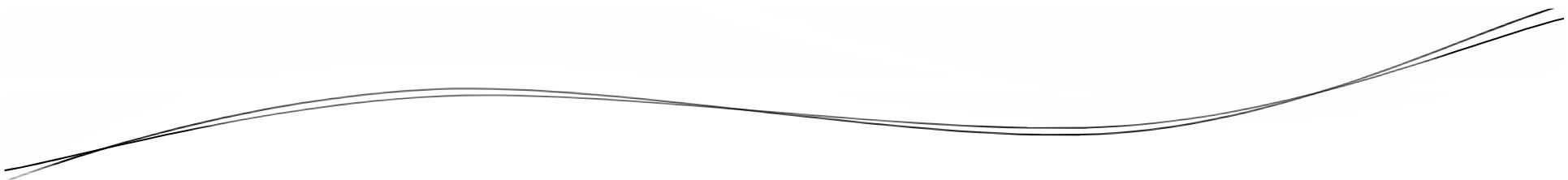
Static Members

- Static methods can access only static members.
- `this` reference is not accessible within static methods.
- `main()` is a static method because it is required to be called without object creation as it's an entry point of the application.



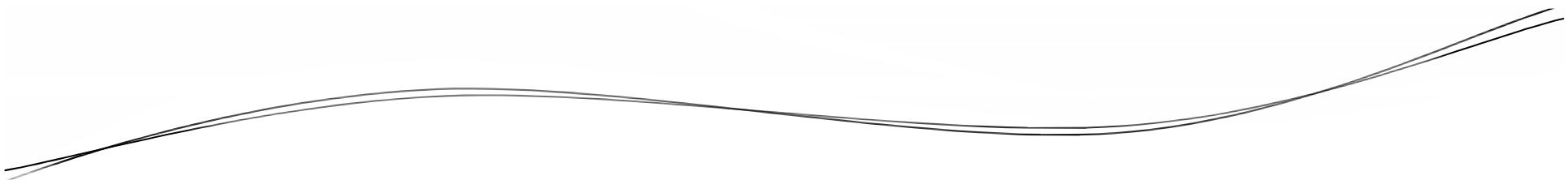
Static Initialization Blocks

By Rahul Barve



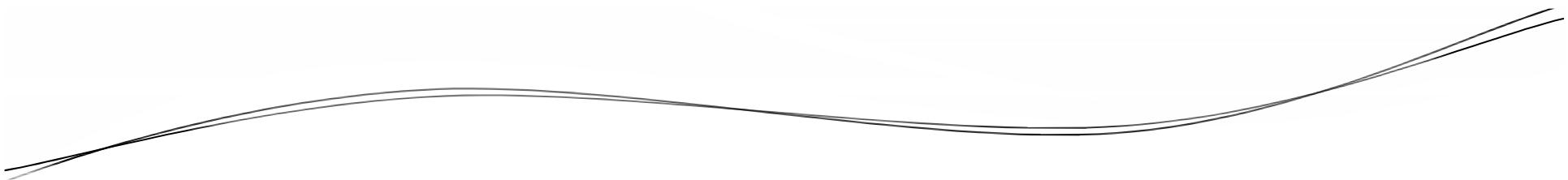
Static Initialization Blocks

- An arbitrary block of code.
- Executes when the class is loaded.
- Used for initializing static variables.



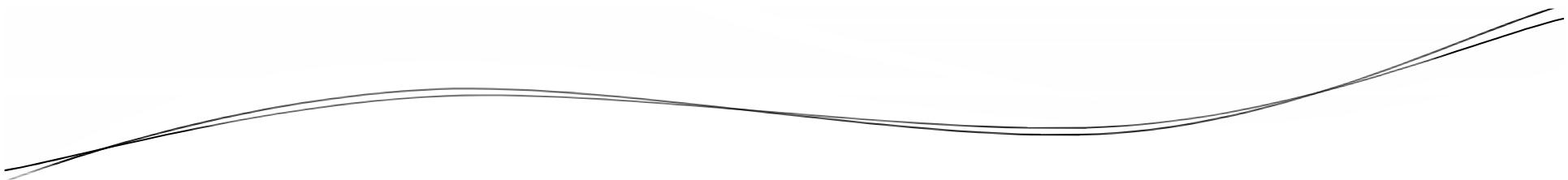
Static Initialization Blocks

```
class MyClass {  
    static {  
        // Some Code  
    }  
}
```



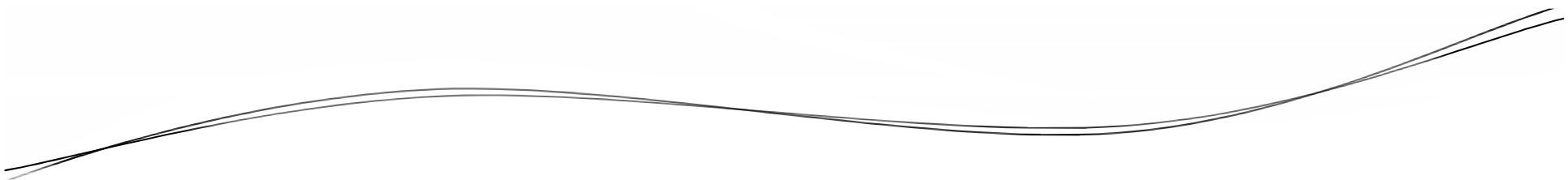
Variables in Java

By Rahul Barve



Variables in Java

- Variables in Java are divided into 3 types:
 - Class Variables (Static Variables)
 - Instance Variables (Non-Static Variables)
 - Local Variables (Must be initialized before usage)



Lets Summarize

- Classes and Objects
- Access Modifiers
- Methods, Overloading of Methods
- Constructors
- this Reference
- Static Members
- Types of Variables

```
1  |
2  public class Book {
3      private String title;
4      private float price;
5      private int pages;
6
7      public Book() { //This is a no-argument constructor
8          //Code to initialize the fields
9          title = "Java Complete Reference";
10         price = 875.75f;
11         pages = 1034;
12     }
13
14
15     public Book(String title, float price, int pages) {
16         //This is a 3 parameterized constructor
17         this.title = title;
18         this.price = price;
19         this.pages = pages;
20     }
21
22
23     public String getTitle() {
24         return title;
25     }
26     public void setTitle(String title) {
27         this.title = title;
28     }
29     public float getPrice() {
30         return price;
```

PrimitiveTypeExample.java | SimpleArrayExample.java | Book.java | BookExample.java | Employee.java | EmployeeExample.java | EmployeeUsingGettersAndSettersExample.java | Flower.java | FlowerUsingStaticExample.java |

```
25     }
26     public void setTitle(String title) {
27         this.title = title;
28     }
29     public float getPrice() {
30         return price;
31     }
32     public void setPrice(float price) {
33         this.price = price;
34     }
35     public int getPages() {
36         return pages;
37     }
38     public void setPages(int pages) {
39         this.pages = pages;
40     }
41
42
43
44
45 }
```

PrimitiveTypeExample.java | SimpleArrayExample.java | Book.java | BookExample.java | Employee.java | EmployeeExample.java | EmployeeUsingGettersAndSettersExample.java | Flower.java | FlowerUsingStaticExample.java |

```
1
2 public class BookExample {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         Book b1 = new Book();
7         Book b2 = new Book("Let us C", 400, 200);
8         //b1.setTitle("Let us C");
9         //b1.setPrice(590.50f);
10        //b1.setPages(300);
11
12        //Get and print the values
13        System.out.println(b1.getTitle());
14        System.out.println(b1.getPrice());
15        System.out.println(b1.getPages());
16
17    }
18
19 }
20
```

```
1  public class Employee {
2      private int empNo;//Makes accessible only within this class; not outside the class
3      private String empName;
4      private float empSalary;
5
6      //Adding a method to assign values for empNo, empName and empSalary
7      public void assignValues(int employeeNo, String employeeName, float employeeSalary) {
8          //Assigning the value available in employeeNo to empNo
9          empNo = employeeNo;
10         //Apply same logic for other assignments
11         empName = employeeName;
12         empSalary = employeeSalary;
13     }
14
15
16     //Adding a method to return the information (String) of the employee
17     public String getEmployeeDetails() {
18         String empDetails =
19             "Employee No: " + empNo + "\nName: " + empName + "\nSalary: " + empSalary;
20         return empDetails;
21     }
22
23     public float getAnnualSalary() {//Method names are camelCased
24         float annualSalary = empSalary * 12;
25         return annualSalary;
26     }
27
28     public boolean isHighlyPaid() {
29         //If an employee earns above 150000 per annum is Highly Paid
30         float annualSalary = getAnnualSalary();
```

```
PrimitiveTypeExample.java | SimpleArrayExample.java | Book.java | BookExample.java | Employee.java | EmployeeExample.java | EmployeeUsingGettersAndSettersExample.java | Flower.java | FlowerUsingStaticExample.java | 
28     public boolean isHighlyPaid() {
29         //If an employee earns above 150000 per annum is Highly Paid
30         float annualSalary = getAnnualSalary();
31         boolean highlyPaid = false;
32         if(annualSalary > 150000)
33             highlyPaid = true;
34         return highlyPaid;
35     }
36
37     public int getEmpNo() {
38         return empNo;
39     }
40
41     public void setEmpNo(int empNo) {
42         this.empNo = empNo;
43     }
44
45     public String getEmpName() {
46         return empName;
47     }
48
49     public void setEmpName(String empName) {
50         this.empName = empName;
51     }
52
53     public float getEmpSalary() {
54         return empSalary;
55     }
56
57     public void setEmpSalary(float empSalary) {
```

PrimitiveTypeExample.java | SimpleArrayExample.java | Book.java | BookExample.java | Employee.java | EmployeeExample.java | EmployeeUsingGettersAndSettersExample.java | Flower.java | FlowerUsingStaticExample.java |

```
55     }
56
57     public void setEmpSalary(float empSalary) {
58         this.empSalary = empSalary;
59     }
60
61
62 }
63
64
65
66
67
68
69
70
71
72 }
```

```
PrimitiveTypeExample.java | SimpleArrayExample.java | Book.java | BookExample.java | Employee.java | EmployeeExample.java | EmployeeUsingGettersAndSettersExample.java | Flower.java | FlowerUsingStaticExample.java | 
1  |
2  public class EmployeeExample {
3  |
4  |     public static void main(String[] args) {
5  |         // TODO Auto-generated method stub
6  |         Employee firstEmployee = new Employee();
7  |         //Assigning values for firstEmployee
8  |         firstEmployee.assignValues(101, "James", 20000.25f);
9  |
10 |         //Retrieving the details of firstEmployee
11 |         String firstEmployeeDetails = firstEmployee.getEmployeeDetails();
12 |         System.out.println(firstEmployeeDetails);
13 |         System.out.println("-----");
14 |         System.out.println(firstEmployee.getEmployeeDetails());
15 |
16 |         float firstEmployeeAnnualSalary = firstEmployee.getAnnualSalary();
17 |         System.out.println("Annual Salary of First Employee: " + firstEmployeeAnnualSalary);
18 |
19 |         boolean highlyPaid = firstEmployee.isHighlyPaid();
20 |         if(highlyPaid)
21 |             System.out.println("This employee is highly paid");
22 |         else
23 |             System.out.println("This employee is not highly paid");
24 |
25 |     }
26 |
27 |
28 }
```

PrimitiveTypeExample.java SimpleArrayExample.java Book.java BookExample.java Employee.java EmployeeExample.java EmployeeUsingGettersAndSettersExample.java Flower.java FlowerUsingStaticExample.java

```
1 public class EmployeeUsingGettersAndSettersExample {
2
3     public static void main(String[] args) {
4         // TODO Auto-generated method stub
5         Employee emp = new Employee();
6         //Assigning the values
7         emp.assignValues(102, "Jack", 50000.75f);
8         //Retrieving the values
9         System.out.println(emp.getEmployeeDetails());
10
11        //Changing the salary of this employee
12        emp.setEmpSalary(75000.80f);
13        //Retrieving the Changed values
14        System.out.println(emp.getEmployeeDetails());
15
16        //Retrieving only the name of the employee
17        String ename = emp.getEmpName();
18        //Retrieving only the salary of the employee
19        float salary = emp.getEmpSalary();
20
21        System.out.println(ename + " is earning a salary : " + salary);
22    }
23
24
25
26
27
28
29
30 }
```

```
PrimitiveTypeExample.java | SimpleArrayExample.java | Book.java | BookExample.java | Employee.java | EmployeeExample.java | EmployeeUsingGettersAndSettersExample.java | Flower.java | FlowerUsingStaticExample.java | 
1  public class Flower {
2      private String name;//Instance Variable
3      private String color;//Instance Variable
4      private static int flowerCount;//Class Variable
5
6      static {
7          flowerCount = 100;
8          System.out.println("Flower class loaded");
9      }
10     public Flower() {
11         // TODO Auto-generated constructor stub
12         flowerCount++;
13     }
14     public Flower(String name, String color) {
15         this.name = name;
16         this.color = color;
17         flowerCount++;
18     }
19
20     static public int getFlowerCount() {//Modifiers may appear in any order
21         return flowerCount;
22     }
23
24     //Remaining code e.g. getters, setters etc
25
26 }
27 }
```

```
1  public class FlowerUsingStaticExample {
2
3      static {
4          System.out.println("Application is about to begin...");
5      }
6
7      static public void main(String[] args) {
8          System.out.println("Application begins...");
9          // TODO Auto-generated method stub
10         Flower f1 = new Flower();
11         Flower f2 = new Flower("Sunflower", "Yellow");
12         Flower f3 = new Flower("Lotus", "Pink");
13
14         System.out.println("Current count of flowers: " + Flower.getFlowerCount());
15         //Adding 5 more flowers
16         for(int a=1;a<=5;a++) {
17             Flower f = new Flower();
18         }
19
20         System.out.println("Latest count of flowers: " + Flower.getFlowerCount());
21
22     }
23
24 }
25
26 }
```

Movie.java X MovieExample.java X Pet.java X PetUsingThisExample.java X Planet.java X PlanetExample.java X

```
1 public class Movie { //Class names are TitleCased
2     String movieId; //Variable names are camelCased
3     String movieTitle;
4     String movieGenre;
5     int movieDuration;
6     //Since no explicit access modifier is used, all the fields have a 'default'
7     //access modifier
8 }
9
10 }
```

```
Movie.java X MovieExample.java X Pet.java X PetUsingThisExample.java X Planet.java X PlanetExample.java X
1
2 public class MovieExample {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         //Creating 2 Movie Objects
7         Movie firstMovie = new Movie();
8         Movie secondMovie = new Movie();
9
10        //Assigning values for firstMovie
11        firstMovie.movieId = "M01";
12        firstMovie.movieTitle = "Life of PI";
13        firstMovie.movieGenre = "Emotional";
14        firstMovie.movieDuration = 125;
15
16        //Apply the same logic for assigning values for secondMovie
17
18        //Retrieving values from firstMovie
19        String firstId = firstMovie.movieId;
20        System.out.println("ID of the first movie: " + firstId);
21        System.out.println("Title of the first movie: " + firstMovie.movieTitle);
22        //Apply the same logic for fetching values of other properties
23
24    }
25
26
27
28
29
30 }
```

```
1 Movie.java X 2 MovieExample.java X 3 Pet.java X 4 PetUsingThisExample.java X 5 Planet.java X 6 PlanetExample.java X
1
2 public class Pet {
3     private String name;
4     private String type;
5     private int age;
6     public Pet() {
7         name = "Alex";
8         type = "Dog";
9         age = 4;
10    }
11    public Pet(String name, String type, int age) {
12        this.name = name;
13        this.type = type;
14        this.age = age;
15    }
16    public String getName() {
17        return name;
18    }
19    public void setName(String name) {
20        this.name = name;
21    }
22    public String getType() {
23        return type;
24    }
25    public void setType(String type) {
26        this.type = type;
27    }
28    public int getAge() {
29        return age;
30    }
31    ...
32 }
```

Movie.java X MovieExample.java X Pet.java X PetUsingThisExample.java X Planet.java X PlanetExample.java X

```
28     public int getAge() {
29         return age;
30     }
31     public void setAge(int age) {
32         this.age = age;
33     }
34     public void assignValues(String nm, String ty, int ag) {
35         //Here variables nm, ty and ag are local variables.
36         //They are not accessible outside the method: assignValues
37         name = nm;
38         type = ty;
39         age = ag;
40     }
41
42     public void assignValuesAgain(String name, String type, int age) {
43         //In this case local variable gets the highest priority
44         //Due to this both the variables are taken from the local scope
45         //To resolve the scope, it is necessary to use 'this' keyword
46         this.name = name;
47         this.type = type;
48         this.age = age;
49     }
50
51     public String getPetInfo() {
52         String petInfo = "Name: " + name + "\nType: " + type + "\nAge: " + age;
53         return petInfo;
54     }
55
56
57 }
```

Movie.java X MovieExample.java X Pet.java X PetUsingThisExample.java X Planet.java X PlanetExample.java X

```
1 public class PetUsingThisExample {
2
3     public static void main(String[] args) {
4         Pet myPet = new Pet();
5         System.out.println("Default Values: \n" + myPet.getPetInfo());
6
7         myPet.assignValuesAgain("Tom", "Cat", 3);
8         System.out.println("Changed Values: \n" + myPet.getPetInfo());
9     }
10
11
12 }
13
```

Movie.java | MovieExample.java | Pet.java | PetUsingThisExample.java | Planet.java | PlanetExample.java

```
1  public class Planet {  
2      //2 Variables  
3      //7 Methods  
4      private String name;  
5      private int moons;  
6      public String getName() {  
7          return name;  
8      }  
9      public void setName(String name) {  
10         this.name = name;  
11     }  
12     public int getMoons() {  
13         return moons;  
14     }  
15     public void setMoons(int moons) {  
16         this.moons = moons;  
17     }  
18     public void assignValues() {//Assigning default values  
19         name = "Earth";  
20         moons = 1;  
21     }  
22     public void assignValues(String pName, int pMoons) {  
23         //This is an overloaded assignValues(String, int)  
24         name = pName;  
25         moons = pMoons;  
26     }  
27     public void assignValues(int pMoons, String pName) {  
28     }
```

Movie.java X MovieExample.java X Pet.java X PetUsingThisExample.java X Planet.java X PlanetExample.java X

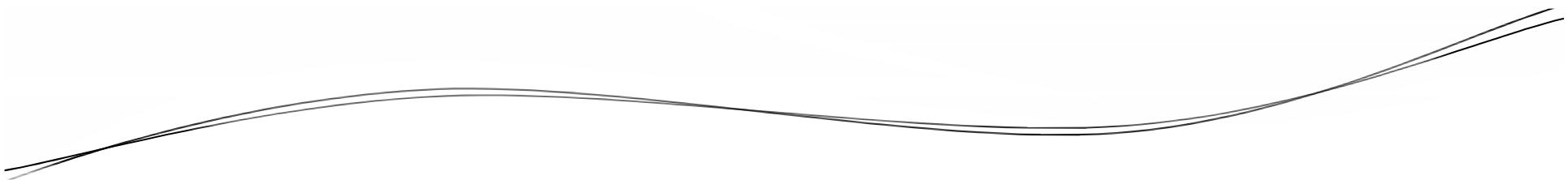
```
22     moons = 1;
23 }
24
25 public void assignValues(String pName, int pMoons) {
26     //This is an overloaded assignValues(String, int)
27     name = pName;
28     moons = pMoons;
29 }
30 public void assignValues(int pMoons, String pName) {
31     //This is an overloaded assignValues(int, String)
32     name = pName;
33     moons = pMoons;
34 }
35
36
37 }
38 }
```

Movie.java X MovieExample.java X Pet.java X PetUsingThisExample.java X Planet.java X PlanetExample.java X

```
1  public class PlanetExample {  
2  
3      public static void main(String[] args) {  
4          // TODO Auto-generated method stub  
5          Planet p1 = new Planet();  
6          Planet p2 = new Planet();  
7          Planet p3 = new Planet();  
8  
9          p1.assignValues(); //Assigning default values  
10         p2.assignValues("Mars", 2);  
11         p3.assignValues(14, "Saturn");  
12  
13         //Remaining code to get and print the values  
14  
15         System.out.println(p2.getName());  
16  
17     }  
18  
19 }  
20  
21 }
```

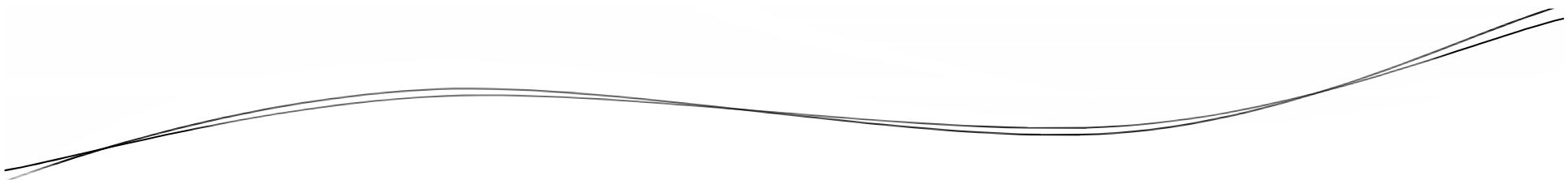
Movie.java X MovieExample.java X Pet.java X PetUsingThisExample.java X Planet.java X PlanetExample.java X Assignment 1.txt X

```
1 Time Limit: 15 Minutes
2
3 Create a class CateringBill with following attributes:
4     int plateCount
5     int ratePerPlate
6
7 Implement getters and setters.
8
9 Implement 2 additional methods as per the following
10    int generateBill() --> Generates bill amount without discount
11
12    int generateBill(int discountPercent) --> Generates bill amount with discount
13
14 Write a main program to test the functionality.
15
16 ***Declare attributes as private
```



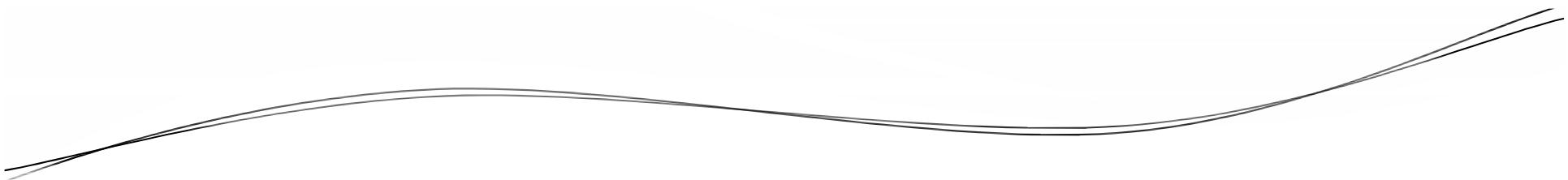
Language Fundamentals

By Rahul Barve



Objectives

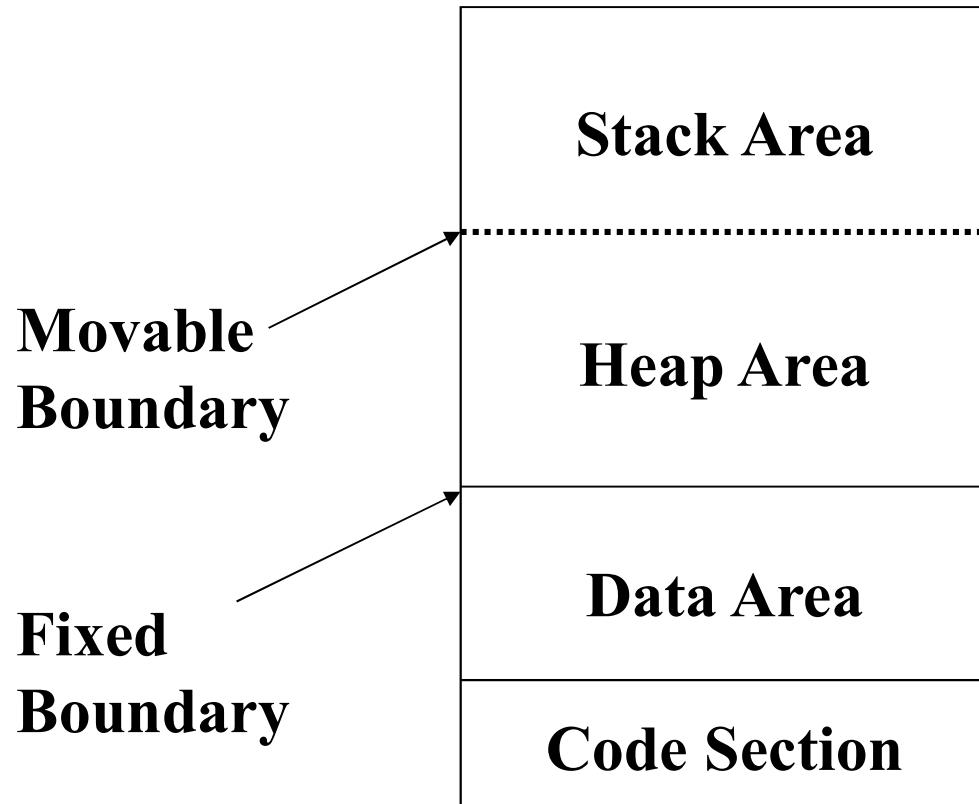
- Memory Mapping
- Parameter Passing
- Working with Arrays.
- Garbage Collection



Memory Mapping

By Rahul Barve

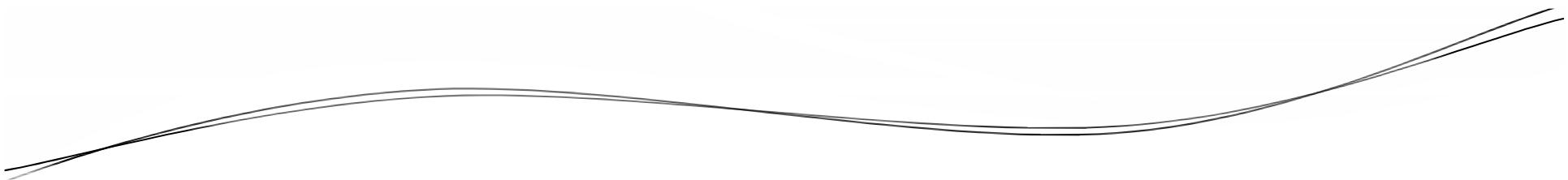
Memory Mapping



Local Variables

Dynamically Allocated Area

Static & Global Variables



Memory Mapping

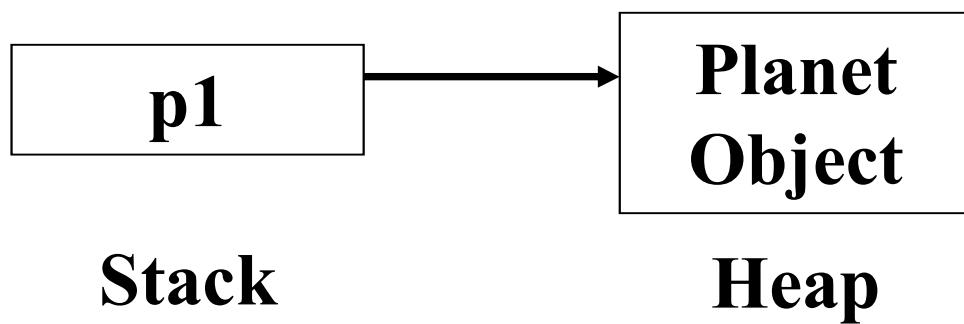
- In Java, object references are created on stack whereas the actual objects are created on heap.
- Java provides a ‘new’ operator that allocates memory dynamically from heap area and returns the reference of the correct type.

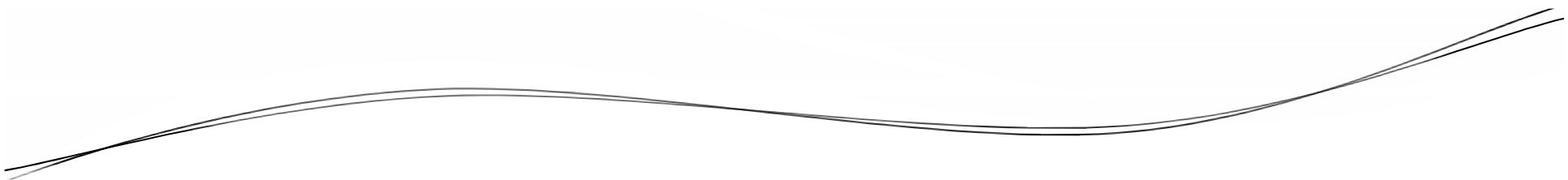
Memory Mapping

- E.g.

```
Planet p1;
```

```
p1 = new Planet();
```





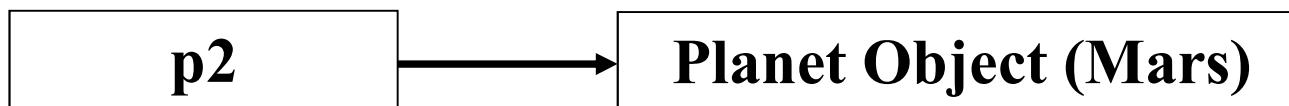
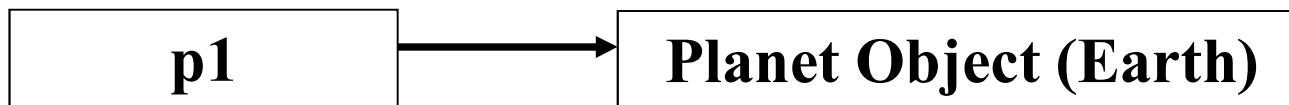
Memory Mapping

- An object may have multiple references but a reference can refer to only one object at a time.

Memory Mapping

```
Planet p1 = new Planet("Earth");
```

```
Planet p2 = new Planet("Mars");
```



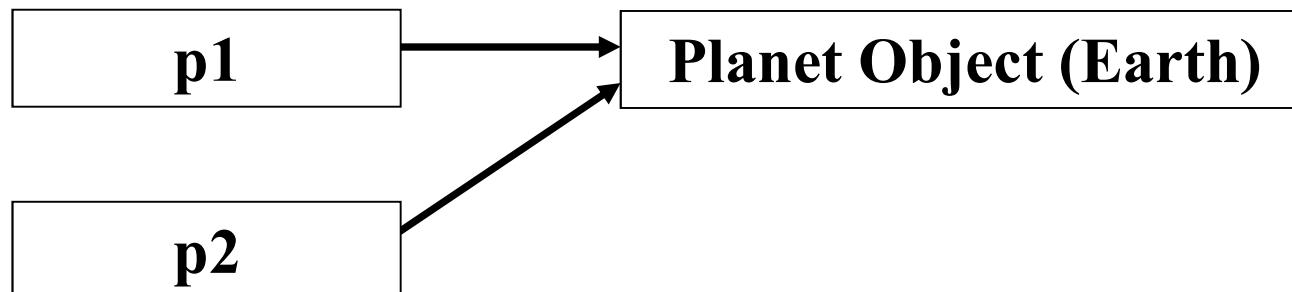
Stack

Heap

Memory Mapping

```
Planet p1 = new Planet("Earth");
```

```
Planet p2 = p1;
```

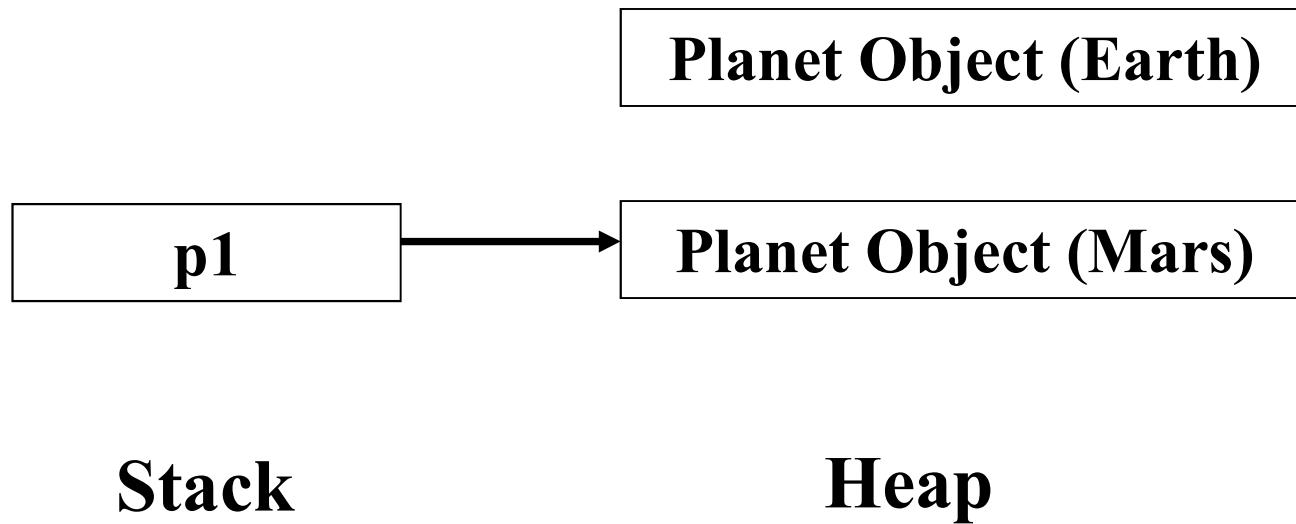


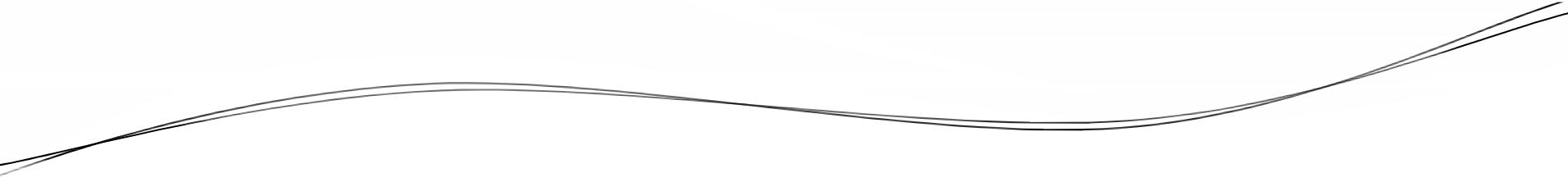
Stack

Heap

Memory Mapping

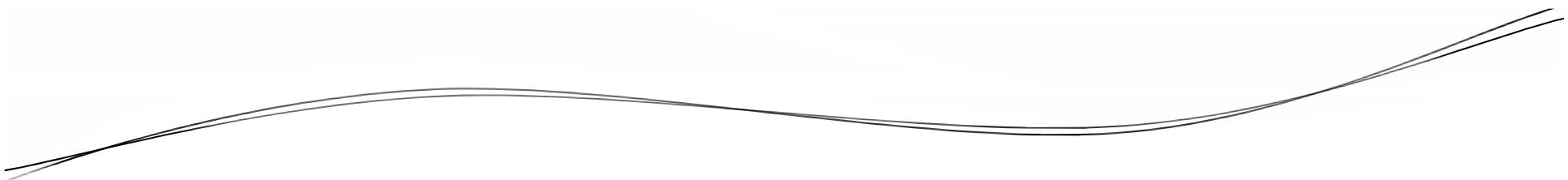
```
Planet p1 = new Planet("Earth");  
p1 = new Planet("Mars");
```





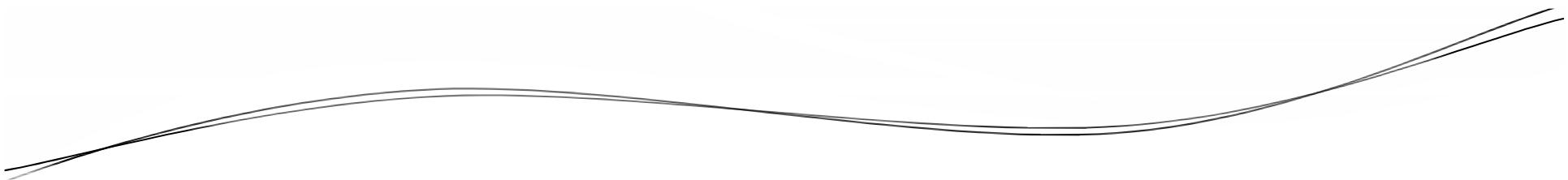
Parameter Passing

By Rahul Barve



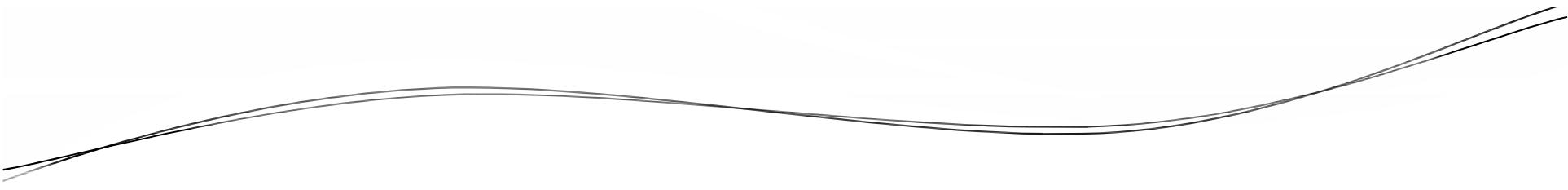
Parameter Passing

- Methods can accept or return parameters either in the form of primitives or object types.
- While passing primitives, a copy of a variable is created on stack and hence primitives are always passed by value.



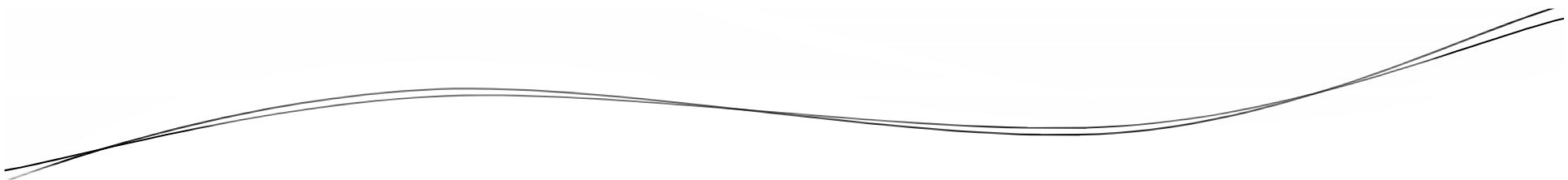
Parameter Passing

- While passing object types, a copy of a reference and not the actual object, is created on stack and hence objects are always passed by reference.



Arrays

By Rahul Barve



Arrays

- Array is a collection of similar typed elements, stored at contiguous memory locations.
- Array has a fixed dimension and the indexing starts from 0.

Arrays

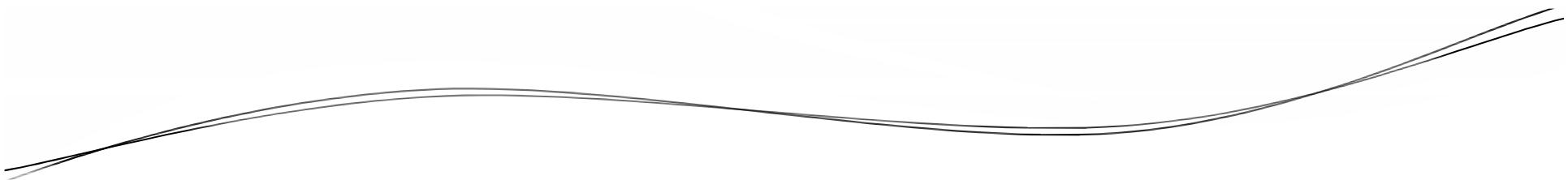
- Arrays can be declared by 2 ways:
 - `int arr[] = new int[5];`
 - `int arr[] = {34, 65, 12};`

Arrays

- Java also provides support for dynamic arrays.
- E.g.

```
int size = 5;
```

```
int arr[] = new int[size];
```



Arrays

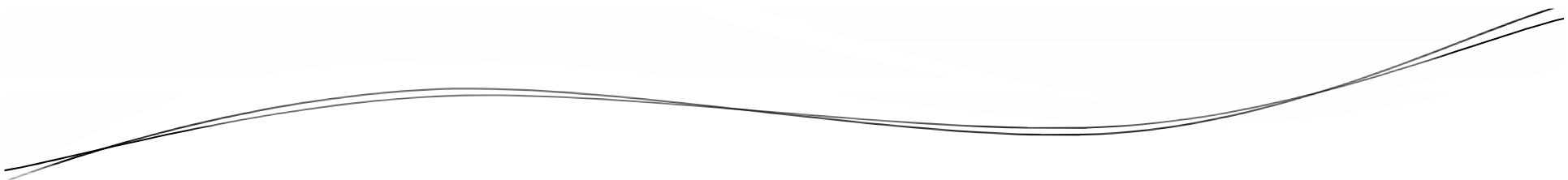
- Every array in Java is treated as an object.
- All array-type objects have a common property called as `length`.

Arrays

```
String skills[] =  
    {"C", "C++", "Java", "SQL", "Python"};  
int size = skills.length;  
  
for(int s=0;s<size;s++) {  
    String skill = skills[s]);  
    System.out.println(skill);  
}
```

OR

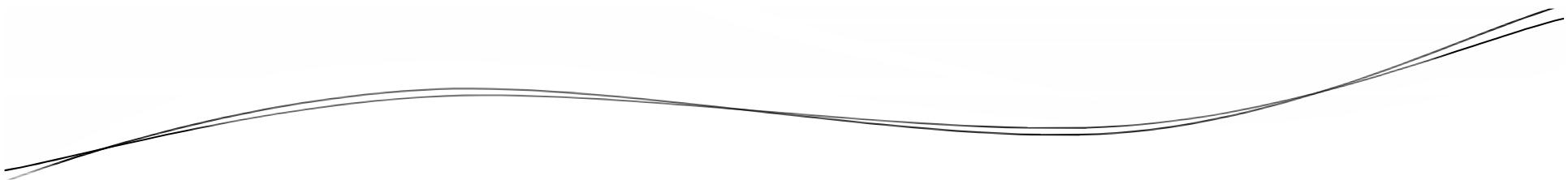
```
for(String skill : skills) {  
    System.out.println(skill);  
}
```



Arrays

- It is also possible to create an array of object types.
- E.g.

```
Planet planets[] = new Planet[2];  
planets[0] = new Planet("Earth");  
planets[1] = new Planet("Mars");
```

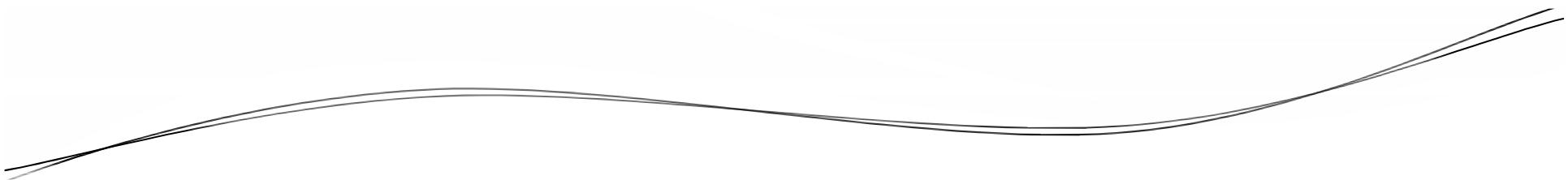


Arrays

```
Planet p1 = new Planet("Earth");
```

```
Planet p2 = new Planet("Mars");
```

```
Planet planets[] = {p1, p2};
```



Arrays

- Like C and C++, Java provides support for two dimensional arrays as well with additional feature as the column dimension can be variant.

Arrays

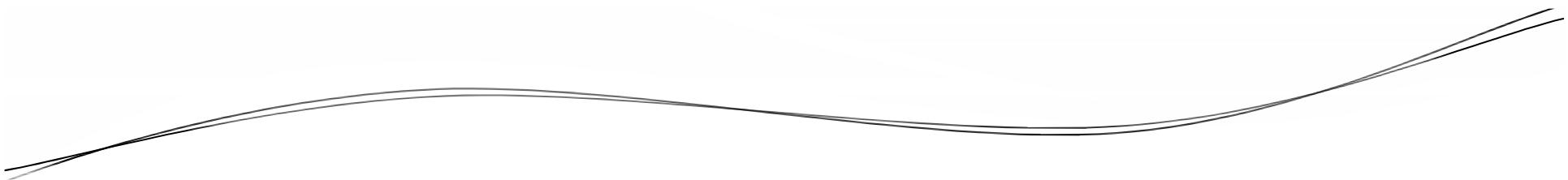
- E.g.

```
int nums[][] = new int[3][];
```

```
nums[0] = new int[1];
```

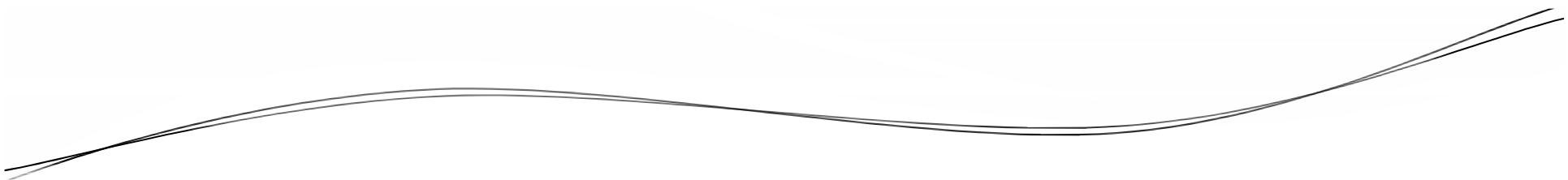
```
nums[1] = new int[2];
```

```
nums[2] = new int[3];
```



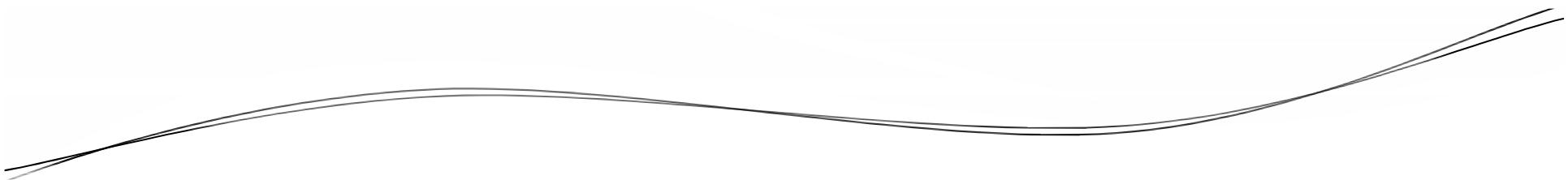
Garbage Collection

By Rahul Barve



Garbage Collection

- Whenever an object has served its purpose, it is no longer needed and hence memory for that object needs to be de-allocated.
- This is taken care by Java Runtime System through a mechanism known as Garbage Collection.

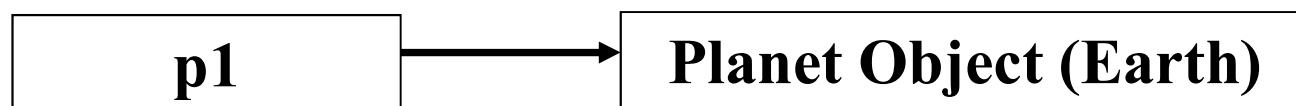


Garbage Collection

- Whenever an object does not have any reference left, the object is marked as unused and becomes eligible for garbage collection.

Garbage Collection

```
Planet p1 = new Planet("Earth");
```

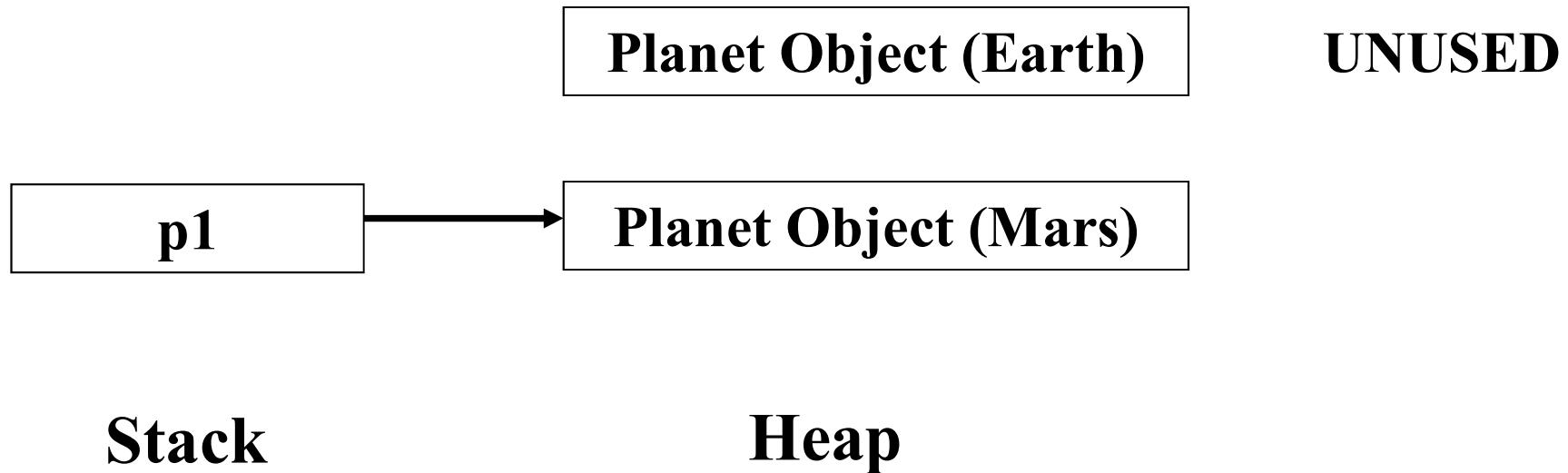


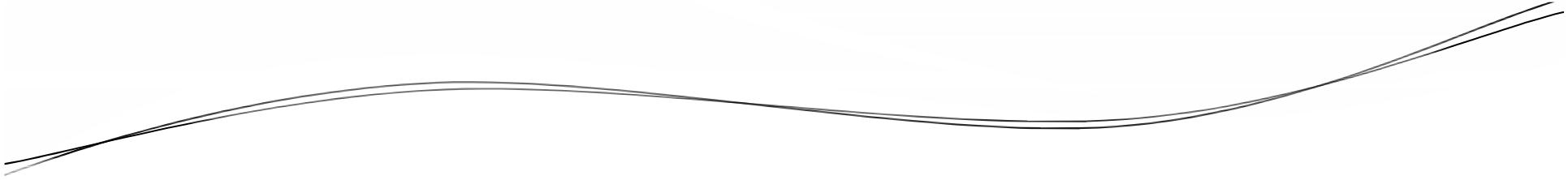
Stack

Heap

Garbage Collection

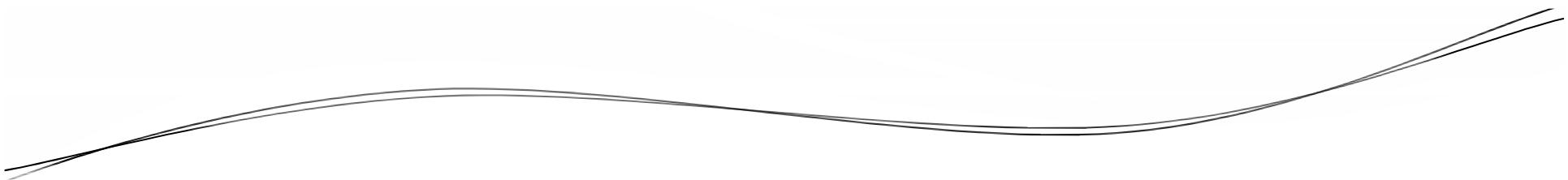
```
Planet p1 = new Planet("Earth");  
p1 = new Planet("Mars");
```





Garbage Collection

- There is a background process known as Garbage Collector which runs periodically, identifies the objects which are marked as UNUSED and de-allocates memory for those objects.



Lets Summarize

- Memory Mapping
- Parameter Passing
- Arrays
- Garbage Collection

```
SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X ContainmentExample.java X
1
2 public class SimpleArrayExample {
3
4     public static void main(String[] args) {
5         // Declaring an array of 3 integers
6         int numbers[] = new int[3];
7         //Initializing the array with some values
8         numbers[0] = 75;
9         numbers[1] = 275;
10        numbers[2] = 175;
11        for(int index=0;index<3;index++) {
12            int num = numbers[index];
13            System.out.println(num);
14        }
15
16        //Declaring and Initializing an array of Strings simultaneously
17        String riverNames[] =
18            {"Ganga", "Yamuna", "Kaveri", "Godavari", "Brahmaputra"};
19        //Obtaining the size of the array
20        int size = riverNames.length;
21        for(int index = 0; index < size; index++)
22            System.out.println(riverNames[index]);
23        System.out.println("-----");
24        //Printing the river names using Enhanced For Loop (For-Each)
25        for(String river : riverNames)
26            System.out.println(river.toUpperCase());
27
28    }
29
30 }
```

```
1
2 public class MemoryMappingExample {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         Employee emp = new Employee();
7         Employee emp2 = new Employee();
8         Employee emp3 = emp2;
9         //In the above statement reference 'emp3' refers to the same object which
10        //is being referred by 'emp2'
11        emp2.setEmpName("James");
12        System.out.println(emp3.getEmpName());
13
14        emp.setEmpName("Jack");
15        System.out.println(emp.getEmpName());
16
17        emp = new Employee();
18        System.out.println(emp.getEmpName());
19
20    }
21
22
23 }
```

```
SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X ContainmentExample.java X
1
2 public class ParameterPassingExample {
3
4     public static void main(String[] args) {
5         int x = 100;
6         System.out.println("Before Change in Primitive: " + x);
7         changeX(x);
8         System.out.println("After Change in Primitive: " + x);
9
10        //-----
11
12         Employee emp = new Employee();
13         emp.assignValues(101, "Tom", 40000);
14         System.out.println("Employee Details before change: " + emp.getEmployeeDetails());
15         changeEmployee(emp);
16         System.out.println("Employee Details after change: " + emp.getEmployeeDetails());
17     }
18
19     private static void changeEmployee(Employee empRef) {
20         empRef.assignValues(102, "Cat", 50000);
21     }
22
23     private static void changeX(int a) {
24         // TODO Auto-generated method stub
25         a = 200;
26         System.out.println("Value inside a method: " + a);
27     }
28
29
30 }
```

```
SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X ContainmentExample.java X
1 public class SimpleArrayAsArgumentExample {
2
3     private static int getSum(int numsArray[]) {
4         int sum = 0;
5         for(int num : numsArray)
6             sum += num;
7         return sum;
8     }
9
10
11    private static double[] getSquareRoots(int numsArray[]) {
12        int size = numsArray.length;//Obtaining size of incoming array
13        //Declaring outgoing array with dimension depending upon the size
14        //of incoming array
15        double squareRootValues[] = new double[size];//Dynamic Array
16        int index = 0;
17        for(int num : numsArray) {
18            double sqRoot = Math.sqrt(num);
19            squareRootValues[index] = sqRoot;
20            index++;
21        }
22        return squareRootValues;
23    }
24
25    public static void main(String[] args) {
26        int numbers[] = {16, 81, 8, 93, 23, 87, 90, 122};
27        int sum = getSum(numbers);
28        System.out.println("Sum = " + sum);
29
30        System.out.println("-----");
31    }
32}
```

SimpleArrayExample.java | MemoryMappingExample.java | ParameterPassingExample.java | SimpleArrayAsArgumentExample.java | ArrayOfObjectsExample.java | Engine.java | MusicSystem.java | Car.java | ContainmentExample.java

```
25     public static void main(String[] args) {
26         int numbers[] = {16, 81, 8, 93, 23, 87, 90, 122};
27         int sum = getSum(numbers);
28         System.out.println("Sum = " + sum);
29
30         System.out.println("-----");
31         double squareRoots[] = getSquareRoots(numbers);
32         for(double sRoot : squareRoots)
33             System.out.println(sRoot);
34
35     }
36
37 }
38 }
```

SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X ContainmentExample.java X

```
1  public class ArrayOfObjectsExample {
2
3      private static String[] getBookNamesWithPagesLessThan1000(Book booksArray) {
4          int size = booksArray.length;
5          String bookNames[] = new String[size];
6          int index = 0;
7          for(Book bk : booksArray) {
8              int pages = bk.getPages();
9              if(pages < 1000) {
10                  String name = bk.getTitle();
11                  bookNames[index] = name;
12                  index++;
13              }
14          }
15      }
16      return bookNames;
17  }
18
19  public static void main(String[] args) {
20      Book books[] = new Book[3];
21      books[0] = new Book("Java", 500, 1000);
22      books[1] = new Book("Python", 475, 800);
23      books[2] = new Book("Angular", 400, 700);
24
25      for(Book bk : books) {
26          System.out.println(bk.getTitle());
27      }
28
29      System.out.println("-----");
30  }
```

SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X ContainmentExample.java X

```
25     for(Book bk : books) {
26         System.out.println(bk.getTitle());
27     }
28
29     System.out.println("-----");
30
31     String names[] = getBookNamesWithPagesLessThan1000(books);
32     for(String name : names) {
33         if(name != null)
34             System.out.println(name);
35     }
36
37 }
38
39 }
40 }
```

```
1  |
2  public class Engine {
3      private String type;//Petrol or Diesel
4      private String power;//1200CC, 1400CC etc
5      public Engine() {
6          type = "Petrol";
7          power = "1200 CC";
8      }
9      public Engine(String type, String power) {
10         this.type = type;
11         this.power = power;
12     }
13     public String getType() {
14         return type;
15     }
16     public void setType(String type) {
17         this.type = type;
18     }
19     public String getPower() {
20         return power;
21     }
22     public void setPower(String power) {
23         this.power = power;
24     }
25
26
27
28 }
```

SimpleArrayExample.java | MemoryMappingExample.java | ParameterPassingExample.java | SimpleArrayAsArgumentExample.java | ArrayOfObjectsExample.java | Engine.java | MusicSystem.java | Car.java | ContainmentExample.java

```
1  public class MusicSystem {
2      private String manufacturer;
3      private String soundEffect;
4      public MusicSystem() {
5          manufacturer = "Panasonic";
6          soundEffect = "Mono";
7      }
8      public MusicSystem(String manufacturer, String soundEffect) {
9          this.manufacturer = manufacturer;
10         this.soundEffect = soundEffect;
11     }
12     public String getManufacturer() {
13         return manufacturer;
14     }
15     public void setManufacturer(String manufacturer) {
16         this.manufacturer = manufacturer;
17     }
18     public String getSoundEffect() {
19         return soundEffect;
20     }
21     public void setSoundEffect(String soundEffect) {
22         this.soundEffect = soundEffect;
23     }
24 }
25
26
27 }
```

```
SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X Containment Example X
1
2 public class Car {
3     private String make;
4     private String model;
5     private Engine engineDetails;//Composition
6     private MusicSystem musicSystemDetails;//Aggregation
7     public Car() {
8         // This is a very basic car model which does not have Music System
9         make = "Maruti";
10        model = "800";
11        //Initializing engine
12        engineDetails = new Engine("Petrol", "800CC");
13        //Not initializing musicSystemDetails because this car does not have that
14    }
15    public Car(String make, String model, Engine engineDetails, MusicSystem musicSystemDetails) {
16        this.make = make;
17        this.model = model;
18        this.engineDetails = engineDetails;
19        this.musicSystemDetails = musicSystemDetails;
20    }
21    public String getMake() {
22        return make;
23    }
24    public void setMake(String make) {
25        this.make = make;
26    }
27    public String getModel() {
28        return model;
29    }
30    public void setModel(String model) {
31    }
}
```

```
28         return model;
29     }
30     public void setModel(String model) {
31         this.model = model;
32     }
33     public Engine getEngineDetails() {
34         return engineDetails;
35     }
36     public void setEngineDetails(Engine engineDetails) {
37         this.engineDetails = engineDetails;
38     }
39     public MusicSystem getMusicSystemDetails() {
40         return musicSystemDetails;
41     }
42     public void setMusicSystemDetails(MusicSystem musicSystemDetails) {
43         this.musicSystemDetails = musicSystemDetails;
44     }
45
46
47
48 }
49 }
```

```
1 MemoryMappingExample.java | ParameterPassingExample.java | SimpleArrayAsArgumentExample.java | ArrayOfObjectsExample.java | Engine.java | MusicSystem.java | Car.java | ContainmentExample.java | new 1
2
3 public class ContainmentExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Car basicCar = new Car(); //This car does not have music system
8
9         //Creating a premium car with music system
10        Engine premiumEngine = new Engine("Diesel", "2600 CC");
11        MusicSystem premiumMusicSystem = new MusicSystem("Sony", "Dolby with Woofers");
12        Car premiumCar =
13            new Car("Toyota", "Innova ZX", premiumEngine, premiumMusicSystem);
14
15         //-----Fetching the Information
16         //Car Model
17         System.out.println("Model of basic car: " + basicCar.getModel());
18         //Car's Engine's Power
19         //This can be done by 2 ways:
20         //1. Explicit Reference Technique
21         //First obtain the Engine associated with the car and then obtain the power
22         Engine basicEngine = basicCar.getEngineDetails();
23         String basicPower = basicEngine.getPower();
24         String fuelType = basicEngine.getType();
25         System.out.println("Power of the engine available in basic car: " + basicPower);
26         System.out.println("Type of the engine available in basic car: " + fuelType);
27         System.out.println("-----");
28         //2. Object Graph Navigation
29         basicPower = basicCar.getEngineDetails().getPower();
30         //int x = basicCar.getEngineDetails().getPower().length();
31         fuelType = basicCar.getEngineDetails().getType();
```

MemoryMappingExample.java | ParameterPassingExample.java | SimpleArrayAsArgumentExample.java | ArrayOfObjectsExample.java | Engine.java | MusicSystem.java | Car.java | ContainmentExample.java | new 1

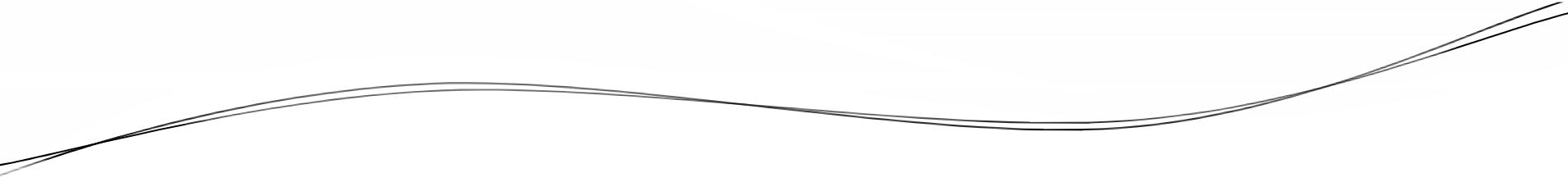
```
25     System.out.println("Type of the engine available in basic car: " + fuelType);
26     System.out.println("-----");
27     //2. Object Graph Navigation
28     basicPower = basicCar.getEngineDetails().getPower();
29     //int x = basicCar.getEngineDetails().getPower().length();
30     fuelType = basicCar.getEngineDetails().getType();
31     System.out.println("Power of the engine available in basic car: " + basicPower);
32     System.out.println("Type of the engine available in basic car: " + fuelType);
33
34     //Fetching the sound effect of the music system of premium car
35     String soundEffect = premiumCar.getMusicSystemDetails().getSoundEffect();
36     System.out.println("Sound effect of music system in premium car: " + soundEffect);
37
38     //Fetching the sound effect of the music system of basic car
39     //soundEffect = basicCar.getMusicSystemDetails().getSoundEffect();
40     //System.out.println("Sound effect of music system in basic car: " + soundEffect);
41     //The above code results into NullPointerException because basicCar's musicSystem is null
42     System.out.println("*****");
43     MusicSystem ms = premiumCar.getMusicSystemDetails();
44     if(ms != null)
45         System.out.println(ms.getSoundEffect());
46
47
48
49 }
50
51 }
52
53 }
54 }
```

ParameterPassingExample.java | SimpleArrayAsArgumentExample.java | ArrayOfObjectsExample.java | Engine.java | MusicSystem.java | Car.java | ContainmentExample.java | new 1 | Assignment 2.txt

```
1 Time Limit: 20 Minutes
2
3 Create a class Account with following attributes:
4     accountNo (int)
5     name (String)
6     balance (float)
7
8 Add following methods in the existing Account class:
9     public void deposit(float)
10    public void withdraw(float)
11    public void transferFunds(Account toAccount, float amountToTransfer)
12
13 Write a main program that builds 2 Account objects and
14 tests the functionality.
15
16 Expectation:
17
18 Account a1 = new Account(..., ..., 100000);
19 Account a2 = new Account(..., ..., 50000);
20
21 a1.transferFunds(a2, 25000);
22
23 a1 ---> 75000
24 a2 ---> 75000
25
26
27
28
29
30
```

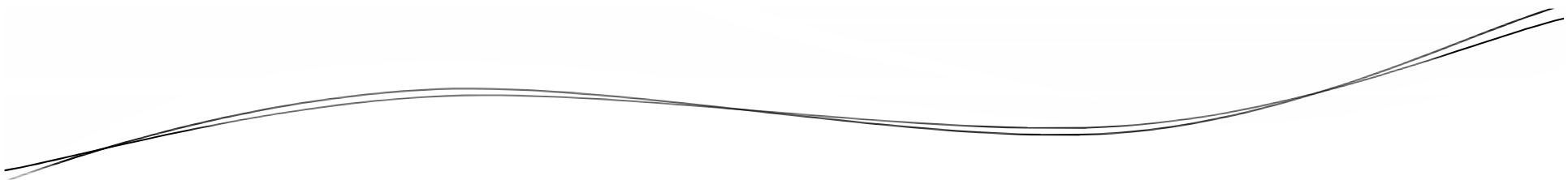
SimpleArrayAsArgumentExample.java ArrayOfObjectsExample.java Engine.java MusicSystem.java Car.java ContainmentExample.java new 1 Assignment 2.txt Assignment 3.txt

```
1 Time Limit: 30 Minutes
2
3 Create a class City with following attributes:
4     String name
5     int population
6
7 Create another class CityProcessor with following methods:
8     public static float getAvgPopulation(City [] )
9     public static String[] getBigCityNames(City [])
10 (Can make use of "!= null" condition)
11
12 The first method must return the average population.
13 The second method must return the names of the big cities.
14 (A city having population > 40 lakhs is a big city)
15
16 Write a main program to test the functionality.
17
18
19
```



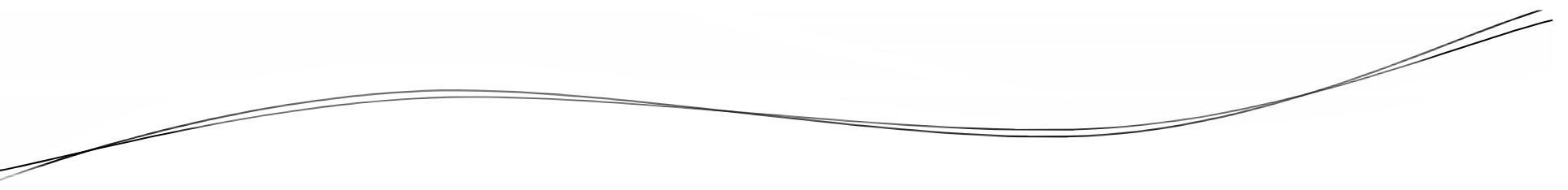
Inheritance

By Rahul Barve



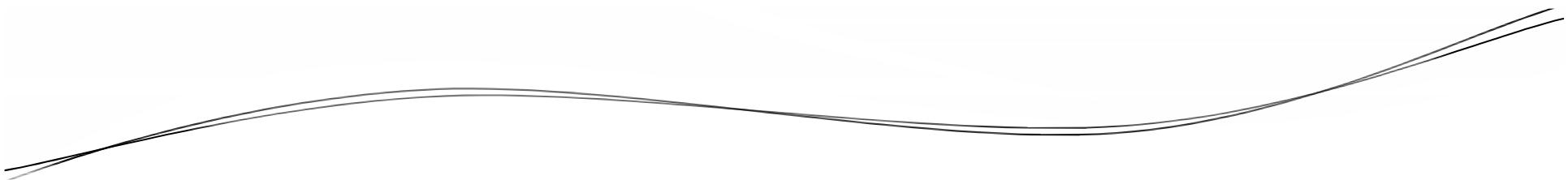
Objectives

- Understanding Containment
- Understanding Inheritance
- Working with super
- Method Overriding
- Abstract Classes
- Introducing final



Containment

By Rahul Barve



Containment

- When an object is composed of another object or several objects then that object model is known as Containment.
- It represents HAS-A relationship.

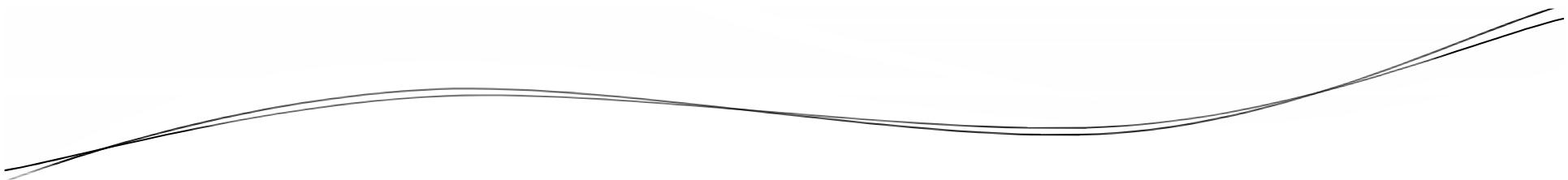
Containment

- A Customer has Address, where Customer is an object that holds another object: Address.
- E.g.

```
class Address {  
    String street, bldg, city;  
    int pincode;  
    ...  
}
```

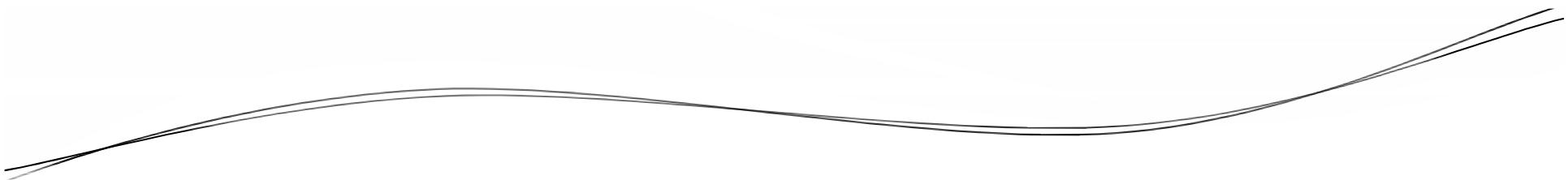
Containment

```
class Customer {  
    String customerId, name, email;  
    Address permanentAddress;  
    ... .  
}
```



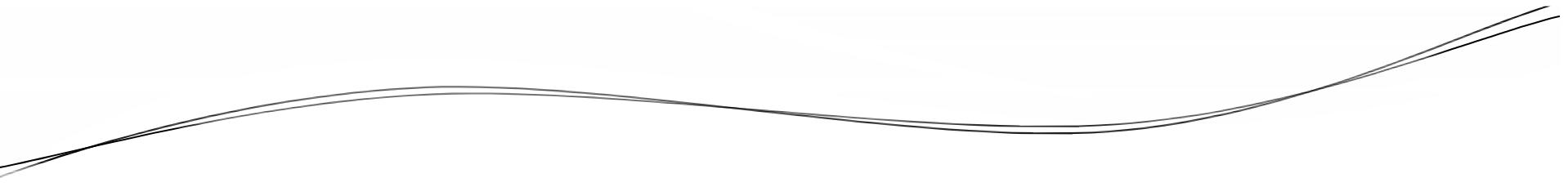
Containment

- In containment, the main object has a dependency on the contained object.
- The dependency is either Strong or Weak.



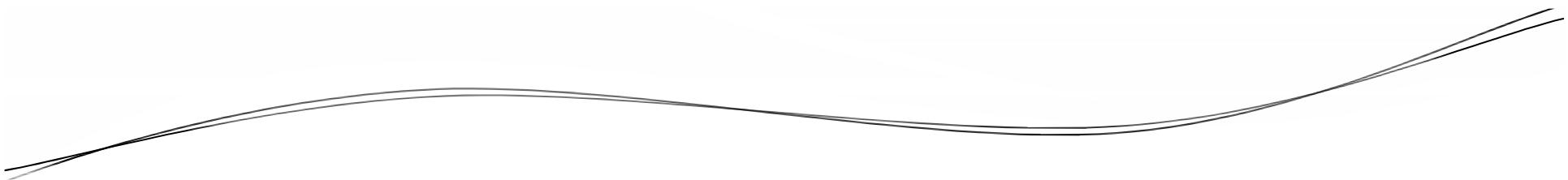
Containment

- Hence containment is of two types:
 - Composition (Strong Dependency)
 - Car has Engine
 - Aggregation (Weak Dependency)
 - Car has MusicSystem



Inheritance

By Rahul Barve



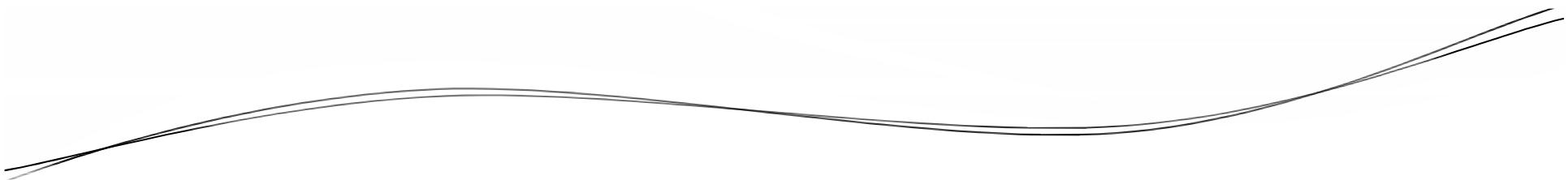
Inheritance

- Inheritance is used to reuse the same structure to adopt different situations.
- It represents IS-A relationship.

Inheritance

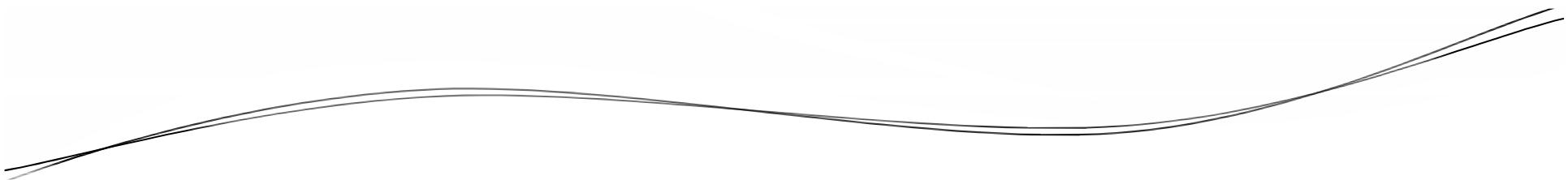
- Every CricketPlayer is a Player, where CricketPlayer is a class which inherits Player class.
- E.g.

```
class Player {  
    ...  
}  
  
class CricketPlayer extends Player {  
    ...  
}
```



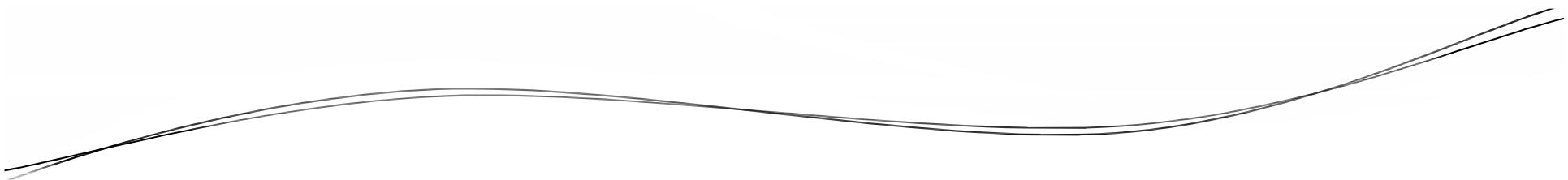
Inheritance

- Java supports 3 types of inheritances:
 - Simple
 - Multilevel
 - Hierarchical
- Inheritance in Java is always public.



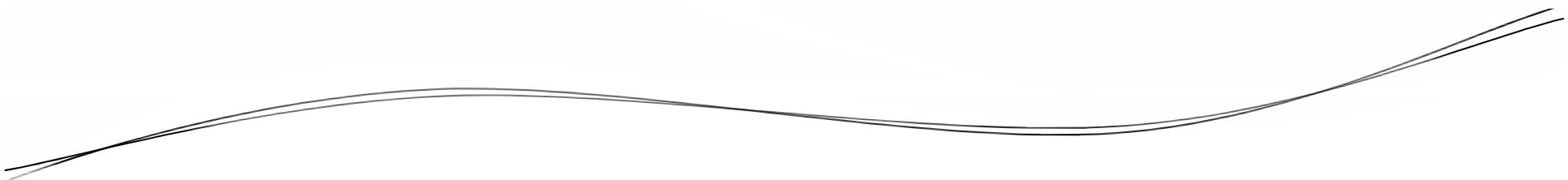
Inheritance

- Constructors in inheritance are invoked from bottom to top and executed from top to bottom.
- In case of parameterized constructors, this is done with the help of super keyword.



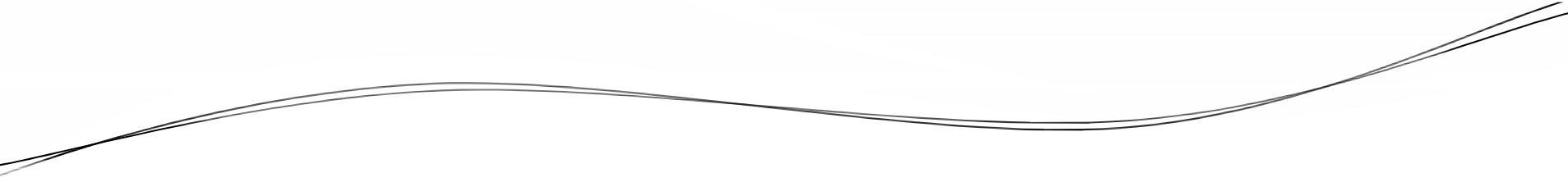
Inheritance

- `super` always refers to an immediate super class.
- Allows to pass the control to the parent class constructor from the child class constructor.



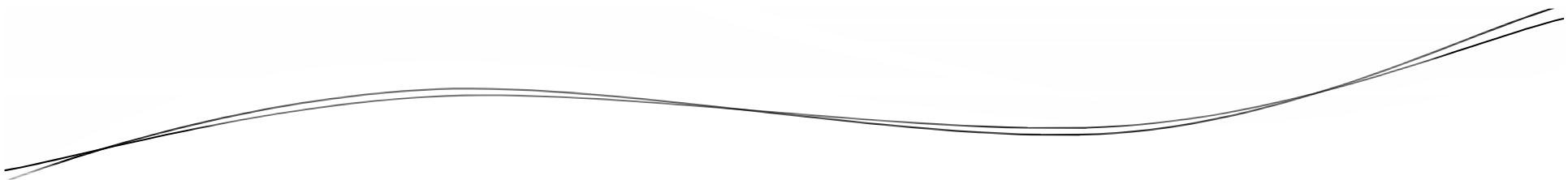
Inheritance

- It must be the first statement in the child class constructor.
- If not used, parent class uses its no-argument constructor.



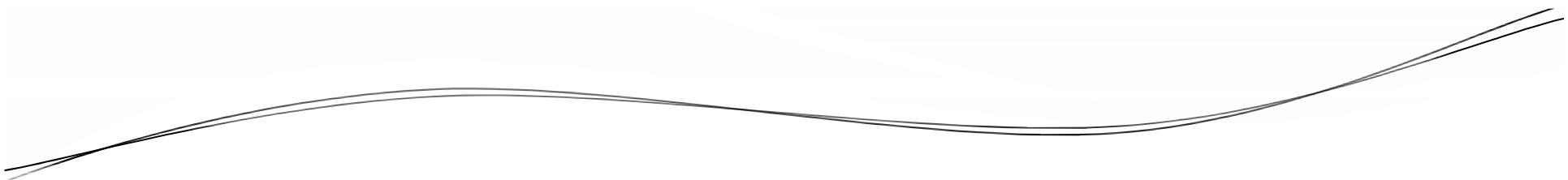
Method Overriding

By Rahul Barve



Method Overriding

- When a subclass defines a method with same name, same signature with identical return type (in case of object references only) as that of the method in the super class, then that method is known as overridden method.



Method Overriding

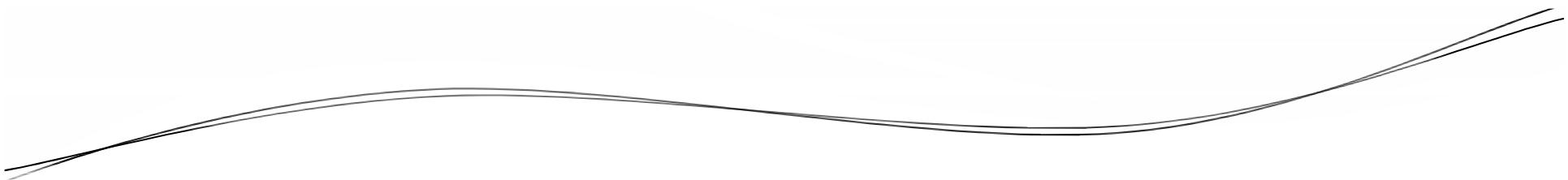
- Overriding is directly related to sub-classing.
- Overriding is required to modify the behavior of a super class's method to suit the new requirement.

Method Overriding

- Consider the hierarchy:
 - Employee ← Manager ← SalesManager and method getSalary () has been overridden.

Method Overriding

- Employee a = new Employee();
a.getSalary() will call Employee's getSalary();
- Manager b = new Manager();
b.getSalary() will call Manager's getSalary();
- SalesManager c = new SalesManager();
c.getSalary() will call SalesManager's getSalary();



Method Overriding

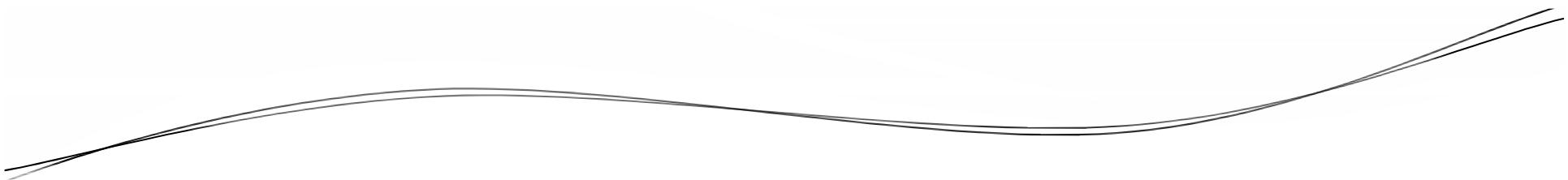
- Dynamic data type governs method selection.
- Every reference has 2 types – static and dynamic.

Method Overriding

- E.g. Employee emp = new Manager();
- In the above statement, the static type of emp is Employee whereas its dynamic type is Manager.
- Hence, emp.getSalary() will invoke Manager's getSalary().

Overloading Vs. Overriding

- Static Polymorphism.
- Method Overloading happens in the same class.
- Signatures of overloaded methods must be different.
- Return types of the overloaded methods may be same or different.
- Dynamic Polymorphism.
- Method Overriding happens between inherited classes.
- Signatures of overridden methods must be same.
- Return types of the overridden methods must be same for primitives and may be identical for objects.



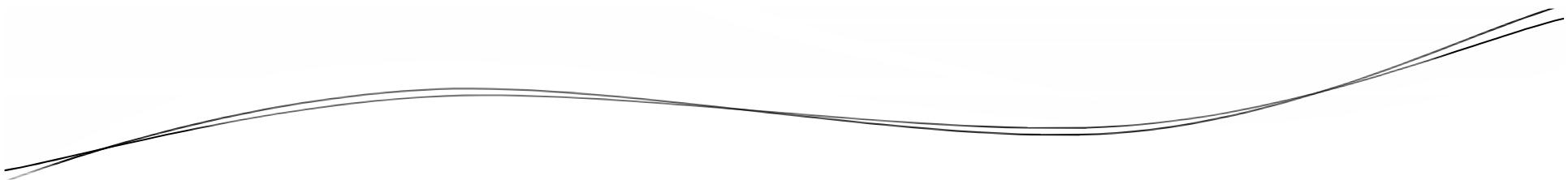
super () Revisited

By Rahul Barve

super() Revisited

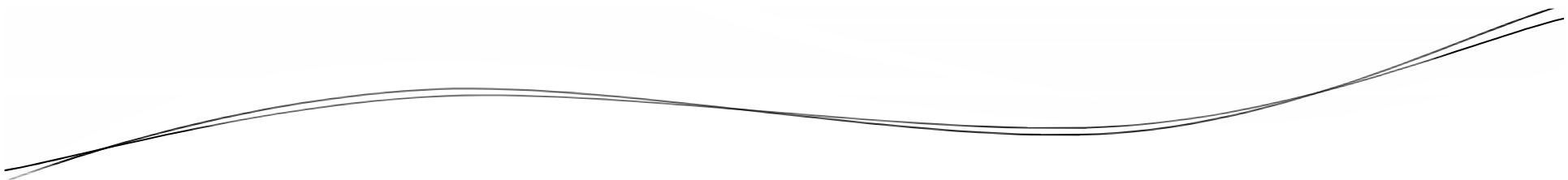
- Can be used to invoke super class method from subclass overridden method.

```
class Derived extends Base{  
    public void myMethod() {  
        super.myMethod();  
        //Invokes myMethod from Base  
    }  
}
```



Abstract Class

By Rahul Barve

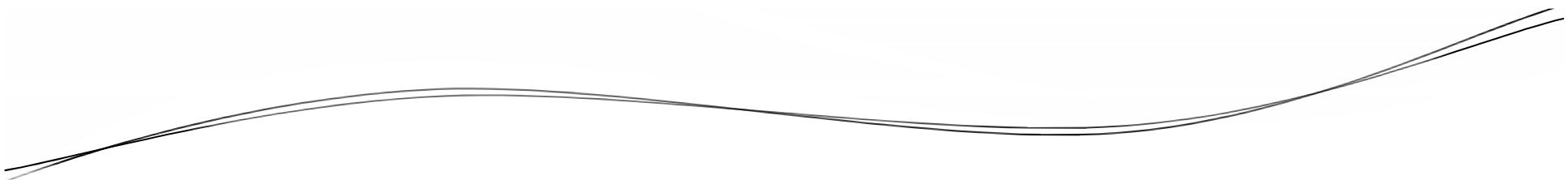


Abstract Class

- Abstract class is a facility to collect generic or common features into a single super class.
- One or more methods are declared but not defined.
- Helps to create base classes that are generic and the implementation is not available.

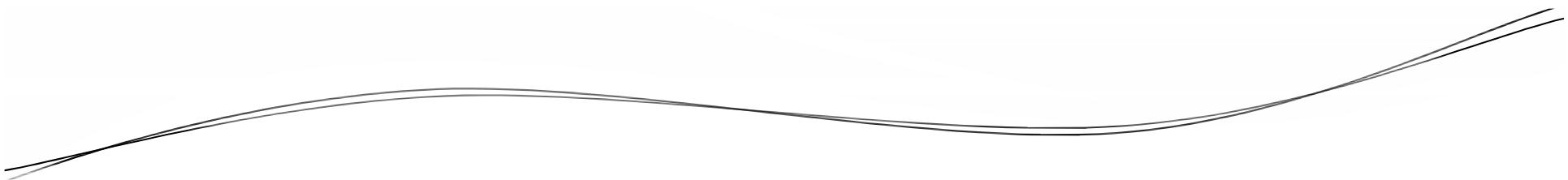
Abstract Class

```
public abstract class Shape {  
    public abstract void draw();  
}  
  
public class Circle extends Shape {  
    public void draw() {  
        //Code to draw Circle  
    }  
}
```



Abstract Class

- Any class that has even one method as abstract, should be declared as abstract.
- Abstract classes cannot be instantiated.
- abstract modifier is not applicable for variables, constructors and static methods.

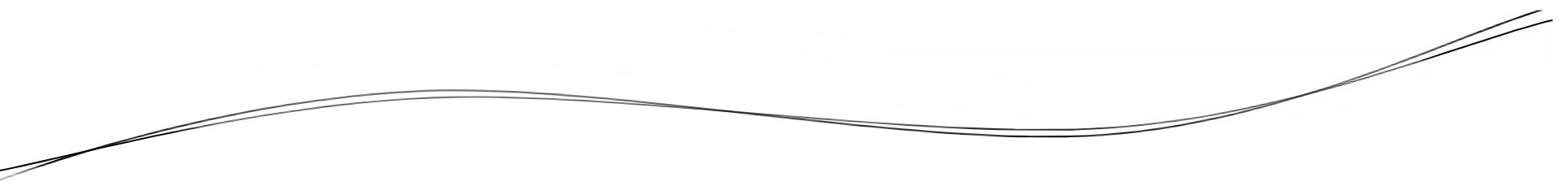


Abstract Class

- Any subclass of an abstract class must implement all the abstract methods; otherwise should be declared as abstract.
- An abstract class may contain concrete methods also.

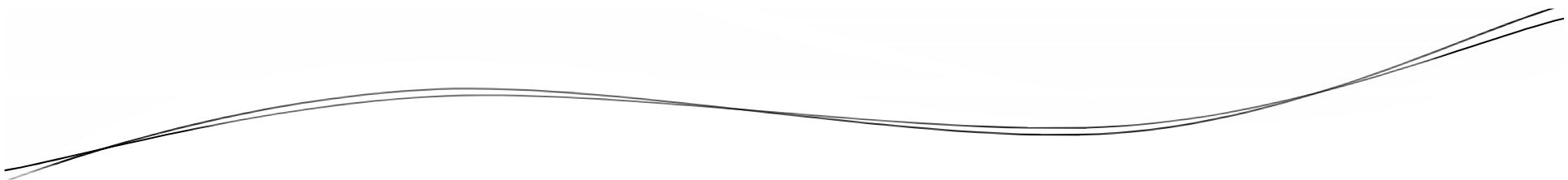
Abstract Class

```
public abstract class Shape {  
    public abstract void draw();  
    public void erase() {  
        //Code to erase the shape  
    }  
}  
public class Circle extends Shape {  
    public void draw() {  
        //Code to draw the Circle  
    }  
}
```



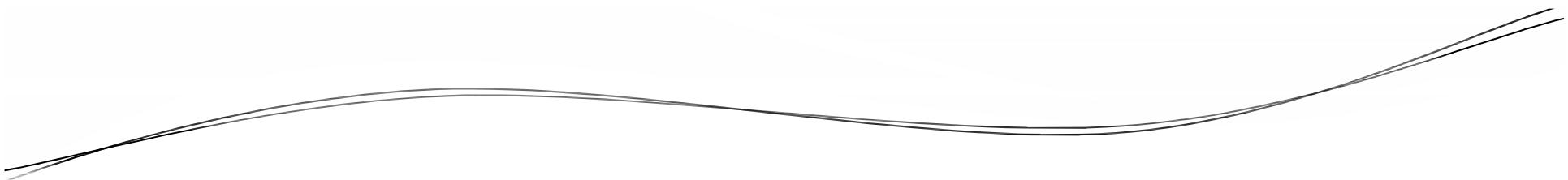
Using **final**

By Rahul Barve



Using **final**

- A keyword in Java.
- Can be used to declare variables that are immutable.



Using **final**

- E.g.

```
final float PI = 3.14f;
```

```
PI = 4.0f; //ERROR
```

- Final variables must be initialized.

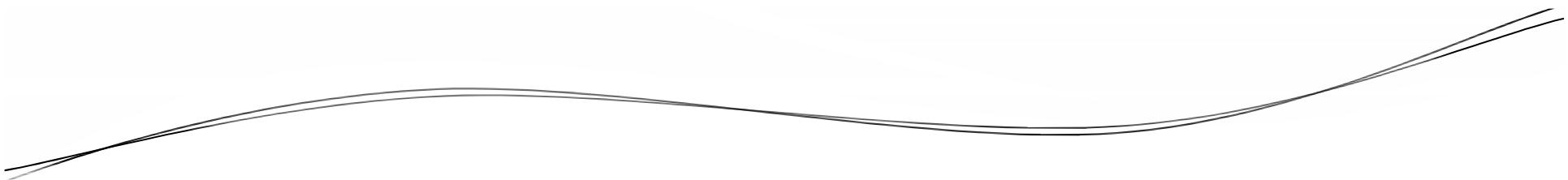
Using **final**

- If a final variable is a reference to an object, it is the reference that must stay the same and not the object.
- E.g.

```
final City c1 = new City("Pune");  
c1.setName("Mumbai"); //OK  
c1 = new City("Mumbai"); //ERROR
```

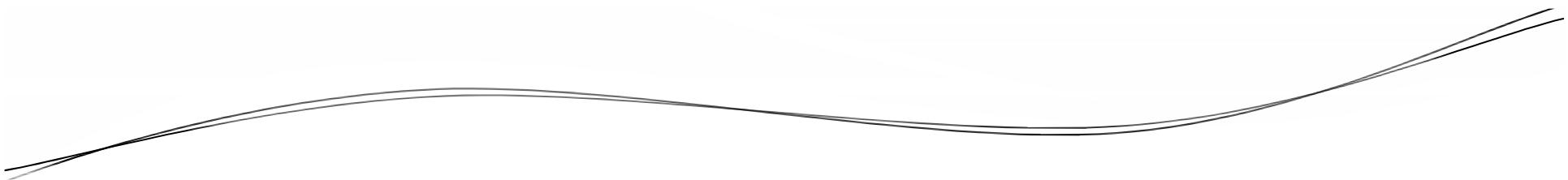
Using **final**

- `final` can also be applied for methods to prevent method overriding.
- `private` methods of a class are implicitly `final`.



Using **final**

- A class can also be declared as final to prevent inheritance.
- If so, all the methods of that class become final.



Lets Summarize

- Containment
- Inheritance
- Working with super
- Method Overriding
- Abstract Classes
- Using final

```
SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X ContainmentExample.java X
1
2 public class SimpleArrayExample {
3
4     public static void main(String[] args) {
5         // Declaring an array of 3 integers
6         int numbers[] = new int[3];
7         //Initializing the array with some values
8         numbers[0] = 75;
9         numbers[1] = 275;
10        numbers[2] = 175;
11        for(int index=0;index<3;index++) {
12            int num = numbers[index];
13            System.out.println(num);
14        }
15
16        //Declaring and Initializing an array of Strings simultaneously
17        String riverNames[] =
18            {"Ganga", "Yamuna", "Kaveri", "Godavari", "Brahmaputra"};
19        //Obtaining the size of the array
20        int size = riverNames.length;
21        for(int index = 0; index < size; index++)
22            System.out.println(riverNames[index]);
23        System.out.println("-----");
24        //Printing the river names using Enhanced For Loop (For-Each)
25        for(String river : riverNames)
26            System.out.println(river.toUpperCase());
27
28    }
29
30 }
```

```
1 public class MemoryMappingExample {  
2     public static void main(String[] args) {  
3         // TODO Auto-generated method stub  
4         Employee emp = new Employee();  
5         Employee emp2 = new Employee();  
6         Employee emp3 = emp2;  
7         //In the above statement reference 'emp3' refers to the same object which  
8         //is being referred by 'emp2'  
9         emp2.setEmpName("James");  
10        System.out.println(emp3.getEmpName());  
11  
12        emp.setEmpName("Jack");  
13        System.out.println(emp.getEmpName());  
14  
15        emp = new Employee();  
16        System.out.println(emp.getEmpName());  
17  
18    }  
19  
20 }  
21  
22  
23 }
```

```
SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X ContainmentExample.java X
1
2 public class ParameterPassingExample {
3
4     public static void main(String[] args) {
5         int x = 100;
6         System.out.println("Before Change in Primitive: " + x);
7         changeX(x);
8         System.out.println("After Change in Primitive: " + x);
9
10        //-----
11
12         Employee emp = new Employee();
13         emp.assignValues(101, "Tom", 40000);
14         System.out.println("Employee Details before change: " + emp.getEmployeeDetails());
15         changeEmployee(emp);
16         System.out.println("Employee Details after change: " + emp.getEmployeeDetails());
17     }
18
19     private static void changeEmployee(Employee empRef) {
20         empRef.assignValues(102, "Cat", 50000);
21     }
22
23     private static void changeX(int a) {
24         // TODO Auto-generated method stub
25         a = 200;
26         System.out.println("Value inside a method: " + a);
27     }
28
29
30 }
```

```
SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X ContainmentExample X
1
2 public class SimpleArrayAsArgumentExample {
3
4     private static int getSum(int numsArray[]) {
5         int sum = 0;
6         for(int num : numsArray)
7             sum += num;
8         return sum;
9     }
10
11    private static double[] getSquareRoots(int numsArray[]) {
12        int size = numsArray.length;//Obtaining size of incoming array
13        //Declaring outgoing array with dimension depending upon the size
14        //of incoming array
15        double squareRootValues[] = new double[size];//Dynamic Array
16        int index = 0;
17        for(int num : numsArray) {
18            double sqRoot = Math.sqrt(num);
19            squareRootValues[index] = sqRoot;
20            index++;
21        }
22        return squareRootValues;
23    }
24
25    public static void main(String[] args) {
26        int numbers[] = {16, 81, 8, 93, 23, 87, 90, 122};
27        int sum = getSum(numbers);
28        System.out.println("Sum = " + sum);
29
30        System.out.println("-----");

```

SimpleArrayExample.java | MemoryMappingExample.java | ParameterPassingExample.java | SimpleArrayAsArgumentExample.java | ArrayOfObjectsExample.java | Engine.java | MusicSystem.java | Car.java | ContainmentExample.java

```
25     public static void main(String[] args) {
26         int numbers[] = {16, 81, 8, 93, 23, 87, 90, 122};
27         int sum = getSum(numbers);
28         System.out.println("Sum = " + sum);
29
30         System.out.println("-----");
31         double squareRoots[] = getSquareRoots(numbers);
32         for(double sRoot : squareRoots)
33             System.out.println(sRoot);
34
35     }
36
37 }
38 }
```

SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X ContainmentExample.java X

```
1  public class ArrayOfObjectsExample {
2
3      private static String[] getBookNamesWithPagesLessThan1000(Book booksArray) {
4          int size = booksArray.length;
5          String bookNames[] = new String[size];
6          int index = 0;
7          for(Book bk : booksArray) {
8              int pages = bk.getPages();
9              if(pages < 1000) {
10                  String name = bk.getTitle();
11                  bookNames[index] = name;
12                  index++;
13              }
14          }
15      }
16      return bookNames;
17  }
18
19  public static void main(String[] args) {
20      Book books[] = new Book[3];
21      books[0] = new Book("Java", 500, 1000);
22      books[1] = new Book("Python", 475, 800);
23      books[2] = new Book("Angular", 400, 700);
24
25      for(Book bk : books) {
26          System.out.println(bk.getTitle());
27      }
28
29      System.out.println("-----");
30  }
```

SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X ContainmentExample.java X

```
25     for(Book bk : books) {
26         System.out.println(bk.getTitle());
27     }
28
29     System.out.println("-----");
30
31     String names[] = getBookNamesWithPagesLessThan1000(books);
32     for(String name : names) {
33         if(name != null)
34             System.out.println(name);
35     }
36
37 }
38
39 }
40 }
```

```
1  |
2  public class Engine {
3      private String type;//Petrol or Diesel
4      private String power;//1200CC, 1400CC etc
5      public Engine() {
6          type = "Petrol";
7          power = "1200 CC";
8      }
9      public Engine(String type, String power) {
10         this.type = type;
11         this.power = power;
12     }
13     public String getType() {
14         return type;
15     }
16     public void setType(String type) {
17         this.type = type;
18     }
19     public String getPower() {
20         return power;
21     }
22     public void setPower(String power) {
23         this.power = power;
24     }
25
26
27
28 }
```

SimpleArrayExample.java | MemoryMappingExample.java | ParameterPassingExample.java | SimpleArrayAsArgumentExample.java | ArrayOfObjectsExample.java | Engine.java | MusicSystem.java | Car.java | ContainmentExample.java

```
1  public class MusicSystem {
2      private String manufacturer;
3      private String soundEffect;
4      public MusicSystem() {
5          manufacturer = "Panasonic";
6          soundEffect = "Mono";
7      }
8      public MusicSystem(String manufacturer, String soundEffect) {
9          this.manufacturer = manufacturer;
10         this.soundEffect = soundEffect;
11     }
12     public String getManufacturer() {
13         return manufacturer;
14     }
15     public void setManufacturer(String manufacturer) {
16         this.manufacturer = manufacturer;
17     }
18     public String getSoundEffect() {
19         return soundEffect;
20     }
21     public void setSoundEffect(String soundEffect) {
22         this.soundEffect = soundEffect;
23     }
24 }
25
26
27 }
```

```
SimpleArrayExample.java X MemoryMappingExample.java X ParameterPassingExample.java X SimpleArrayAsArgumentExample.java X ArrayOfObjectsExample.java X Engine.java X MusicSystem.java X Car.java X Containment Example X
1
2 public class Car {
3     private String make;
4     private String model;
5     private Engine engineDetails;//Composition
6     private MusicSystem musicSystemDetails;//Aggregation
7     public Car() {
8         // This is a very basic car model which does not have Music System
9         make = "Maruti";
10        model = "800";
11        //Initializing engine
12        engineDetails = new Engine("Petrol", "800CC");
13        //Not initializing musicSystemDetails because this car does not have that
14    }
15    public Car(String make, String model, Engine engineDetails, MusicSystem musicSystemDetails) {
16        this.make = make;
17        this.model = model;
18        this.engineDetails = engineDetails;
19        this.musicSystemDetails = musicSystemDetails;
20    }
21    public String getMake() {
22        return make;
23    }
24    public void setMake(String make) {
25        this.make = make;
26    }
27    public String getModel() {
28        return model;
29    }
30    public void setModel(String model) {
```

```
28         return model;
29     }
30     public void setModel(String model) {
31         this.model = model;
32     }
33     public Engine getEngineDetails() {
34         return engineDetails;
35     }
36     public void setEngineDetails(Engine engineDetails) {
37         this.engineDetails = engineDetails;
38     }
39     public MusicSystem getMusicSystemDetails() {
40         return musicSystemDetails;
41     }
42     public void setMusicSystemDetails(MusicSystem musicSystemDetails) {
43         this.musicSystemDetails = musicSystemDetails;
44     }
45
46
47
48 }
49 }
```

```
1 MemoryMappingExample.java | ParameterPassingExample.java | SimpleArrayAsArgumentExample.java | ArrayOfObjectsExample.java | Engine.java | MusicSystem.java | Car.java | ContainmentExample.java | new 1
2
3 public class ContainmentExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Car basicCar = new Car(); //This car does not have music system
8
9         //Creating a premium car with music system
10        Engine premiumEngine = new Engine("Diesel", "2600 CC");
11        MusicSystem premiumMusicSystem = new MusicSystem("Sony", "Dolby with Woofers");
12        Car premiumCar =
13            new Car("Toyota", "Innova ZX", premiumEngine, premiumMusicSystem);
14
15         //-----Fetching the Information
16         //Car Model
17         System.out.println("Model of basic car: " + basicCar.getModel());
18         //Car's Engine's Power
19         //This can be done by 2 ways:
20         //1. Explicit Reference Technique
21         //First obtain the Engine associated with the car and then obtain the power
22         Engine basicEngine = basicCar.getEngineDetails();
23         String basicPower = basicEngine.getPower();
24         String fuelType = basicEngine.getType();
25         System.out.println("Power of the engine available in basic car: " + basicPower);
26         System.out.println("Type of the engine available in basic car: " + fuelType);
27         System.out.println("-----");
28         //2. Object Graph Navigation
29         basicPower = basicCar.getEngineDetails().getPower();
30         //int x = basicCar.getEngineDetails().getPower().length();
31         fuelType = basicCar.getEngineDetails().getType();
```

MemoryMappingExample.java | ParameterPassingExample.java | SimpleArrayAsArgumentExample.java | ArrayOfObjectsExample.java | Engine.java | MusicSystem.java | Car.java | ContainmentExample.java | new 1

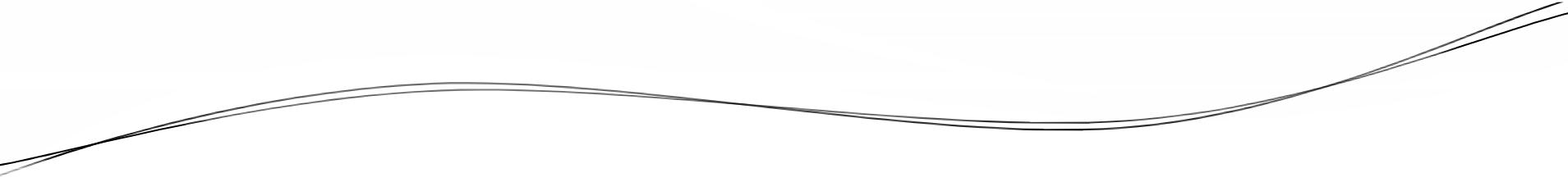
```
25     System.out.println("Type of the engine available in basic car: " + fuelType);
26     System.out.println("-----");
27     //2. Object Graph Navigation
28     basicPower = basicCar.getEngineDetails().getPower();
29     //int x = basicCar.getEngineDetails().getPower().length();
30     fuelType = basicCar.getEngineDetails().getType();
31     System.out.println("Power of the engine available in basic car: " + basicPower);
32     System.out.println("Type of the engine available in basic car: " + fuelType);
33
34     //Fetching the sound effect of the music system of premium car
35     String soundEffect = premiumCar.getMusicSystemDetails().getSoundEffect();
36     System.out.println("Sound effect of music system in premium car: " + soundEffect);
37
38     //Fetching the sound effect of the music system of basic car
39     //soundEffect = basicCar.getMusicSystemDetails().getSoundEffect();
40     //System.out.println("Sound effect of music system in basic car: " + soundEffect);
41     //The above code results into NullPointerException because basicCar's musicSystem is null
42     System.out.println("*****");
43     MusicSystem ms = premiumCar.getMusicSystemDetails();
44     if(ms != null)
45         System.out.println(ms.getSoundEffect());
46
47
48
49 }
50
51 }
52
53 }
54 }
```

ParameterPassingExample.java | SimpleArrayAsArgumentExample.java | ArrayOfObjectsExample.java | Engine.java | MusicSystem.java | Car.java | ContainmentExample.java | new 1 | Assignment 2.txt

```
1 Time Limit: 20 Minutes
2
3 Create a class Account with following attributes:
4     accountNo (int)
5     name (String)
6     balance (float)
7
8 Add following methods in the existing Account class:
9     public void deposit(float)
10    public void withdraw(float)
11    public void transferFunds(Account toAccount, float amountToTransfer)
12
13 Write a main program that builds 2 Account objects and
14 tests the functionality.
15
16 Expectation:
17
18 Account a1 = new Account(..., ..., 100000);
19 Account a2 = new Account(..., ..., 50000);
20
21 a1.transferFunds(a2, 25000);
22
23 a1 ---> 75000
24 a2 ---> 75000
25
26
27
28
29
30
```

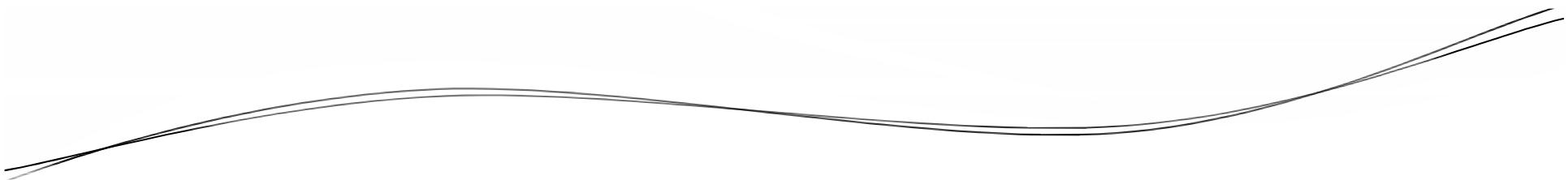
SimpleArrayAsArgumentExample.java ArrayOfObjectsExample.java Engine.java MusicSystem.java Car.java ContainmentExample.java new 1 Assignment 2.txt Assignment 3.txt

```
1 Time Limit: 30 Minutes
2
3 Create a class City with following attributes:
4     String name
5     int population
6
7 Create another class CityProcessor with following methods:
8     public static float getAvgPopulation(City [] )
9     public static String[] getBigCityNames(City [])
10 (Can make use of "!= null" condition)
11
12 The first method must return the average population.
13 The second method must return the names of the big cities.
14 (A city having population > 40 lakhs is a big city)
15
16 Write a main program to test the functionality.
17
18
19
```



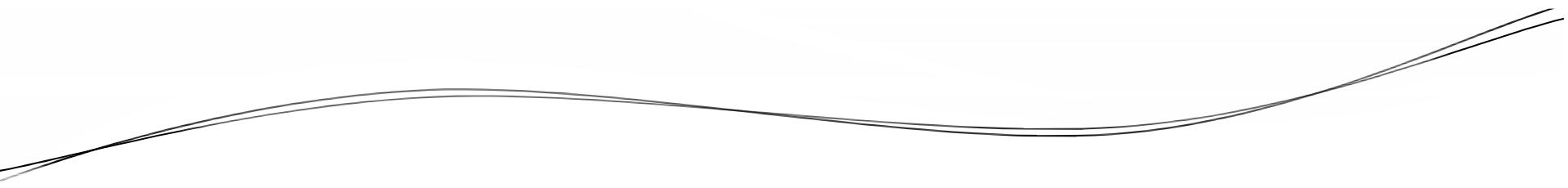
Packages

By Rahul Barve



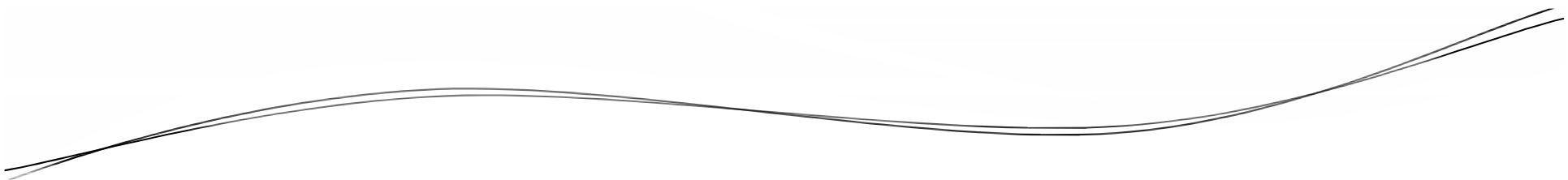
Objectives

- Understanding Packages
- Need for Packages
- Access Modifiers Revisited
- Exploring `java.lang`



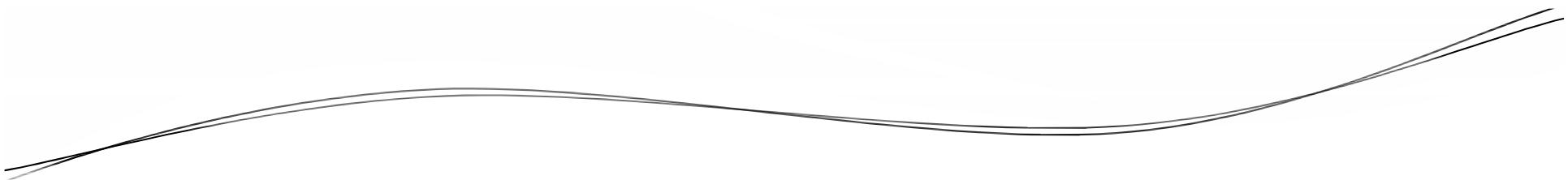
Package

By Rahul Barve



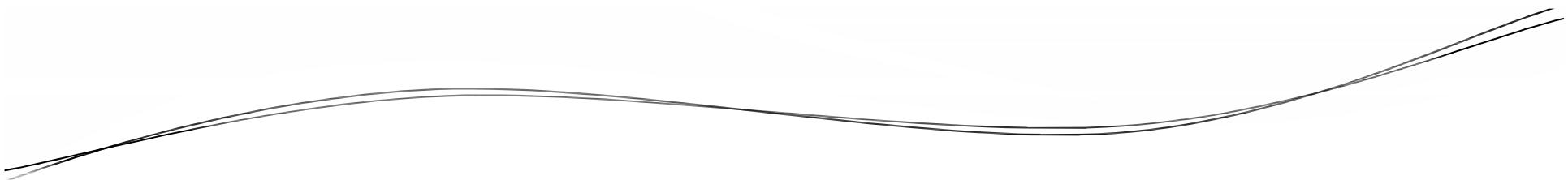
Package

- Package is a collection of classes and interfaces.
- Used to keep class library isolated from other libraries.
- Can be used to reduce naming conflicts about classes and interfaces.



Creating a Package

By Rahul Barve



Creating a Package

- Packages are created using package statement.
- If used, it must be the first statement in the Java source file.

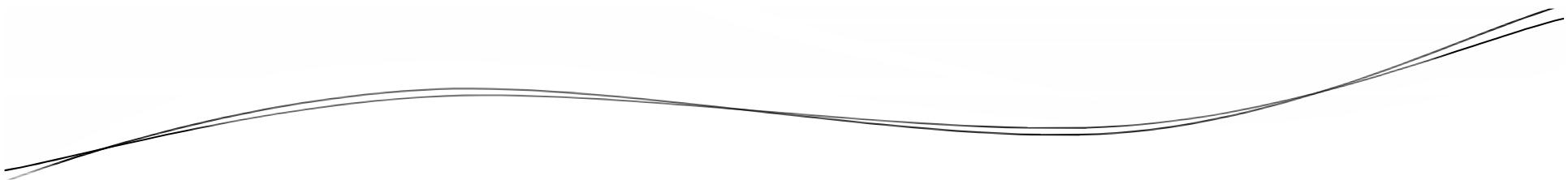
Creating a Package

- Syntax:

```
package <package-name>;  
    //class definition
```

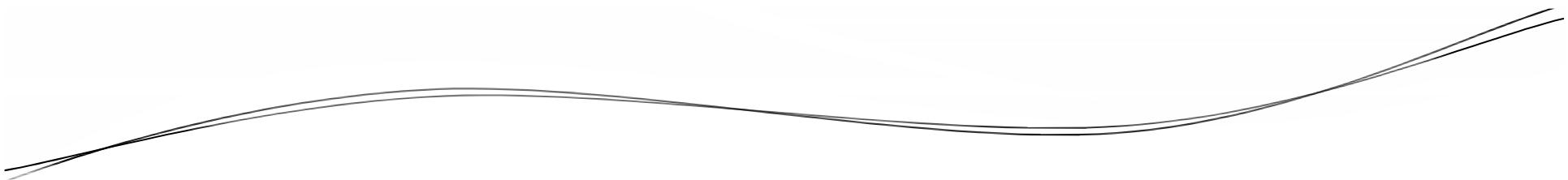
- E.g.

```
package test;  
public class Test {  
    ...  
}
```



Accessing Classes

- If two or multiple classes are belonging to same package, one class can directly access other classes irrespective of whether they are declared as public or not.

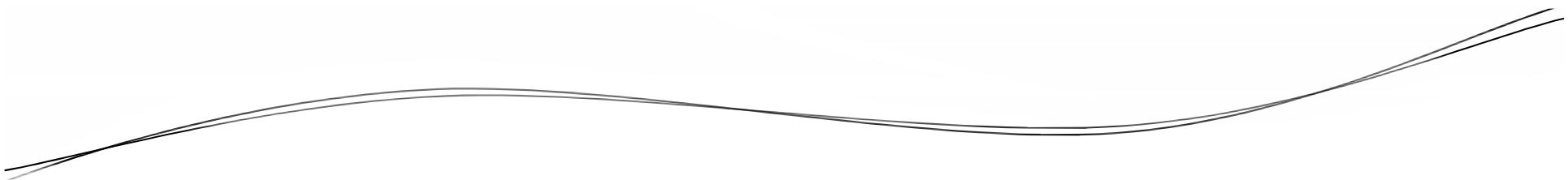


Accessing Classes

- E.g.

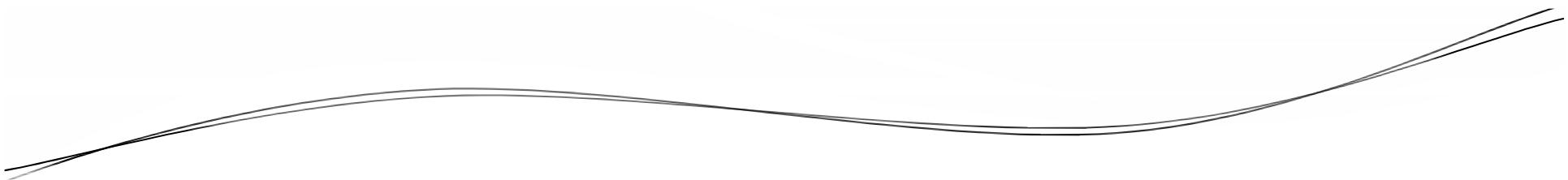
```
package business;  
public class Address { ... }
```

```
package business;  
public class Customer {  
    Address commAddress;  
  
    ...  
}
```



Accessing Classes

- If classes belong to different packages, then one class has to import classes from other packages provided they are declared as public.
- This is done using the `import` statement.

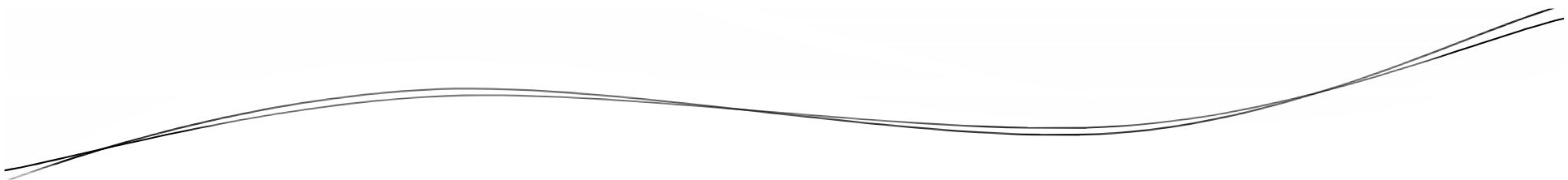


Accessing Classes

- E.g.

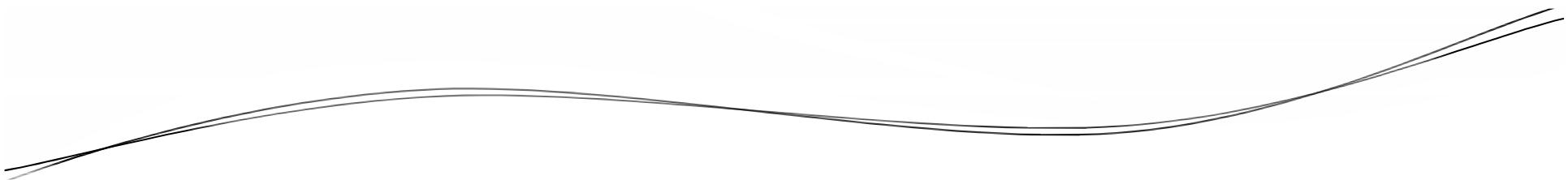
```
package residence;  
public class Address { ... }
```

```
package college;  
import residence.Address;  
public class Student {  
    Address residentialAddress;  
    ...  
}
```



Sub Packages

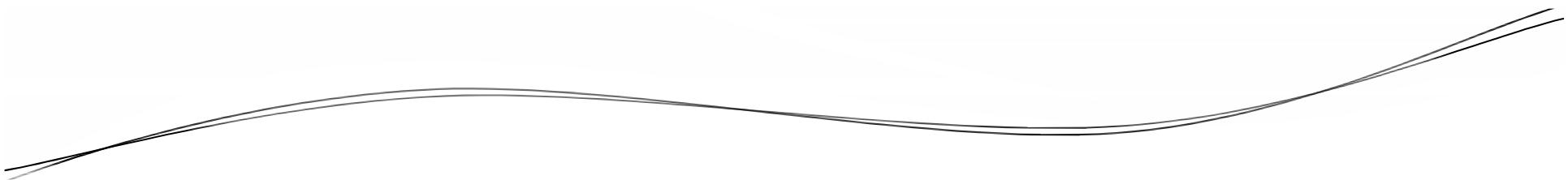
By Rahul Barve



Sub Packages

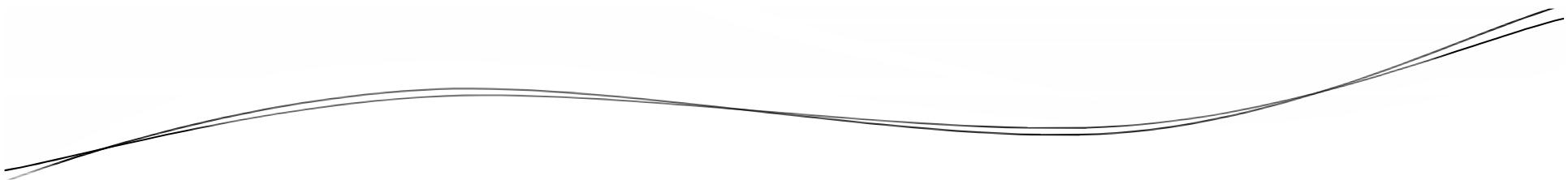
- A package within another package is called as a sub package.
- To import classes from a sub package, the name of the super package is mandatory.
- E.g.

```
import p1.p2.*;
```



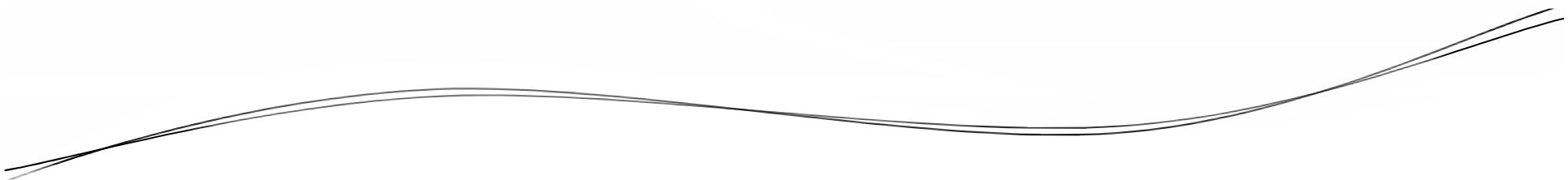
Default Package

By Rahul Barve



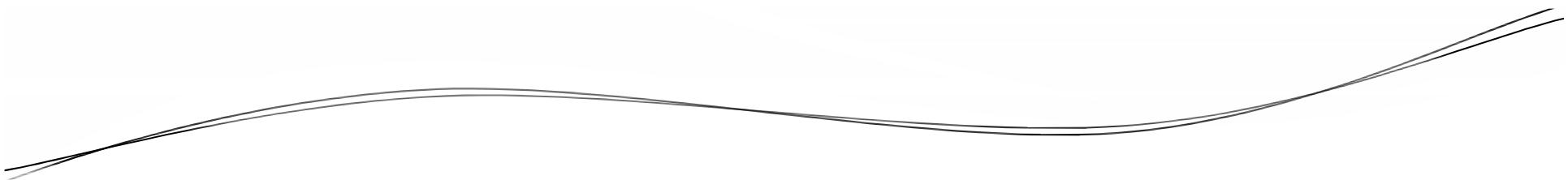
Default Package

- Whenever a class is declared without package statement, then that class is said to be a part of a default package.



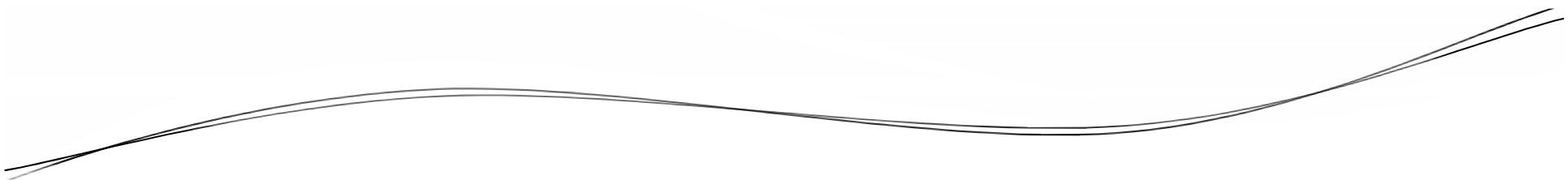
Default Package

- Such classes cannot be imported by classes coming from other packages; and hence the use of default package is discouraged.



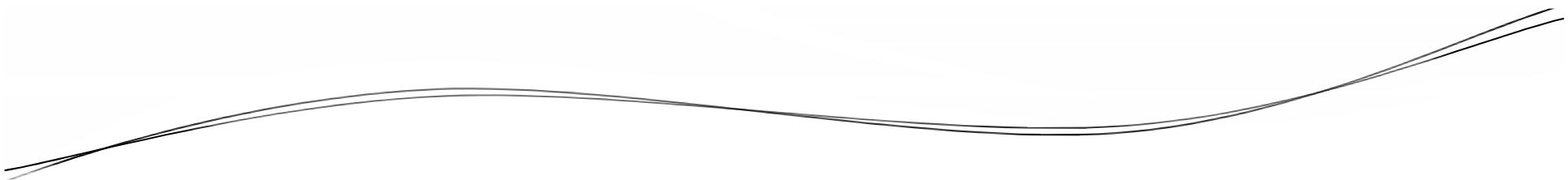
Access Modifiers Revisited

By Rahul Barve



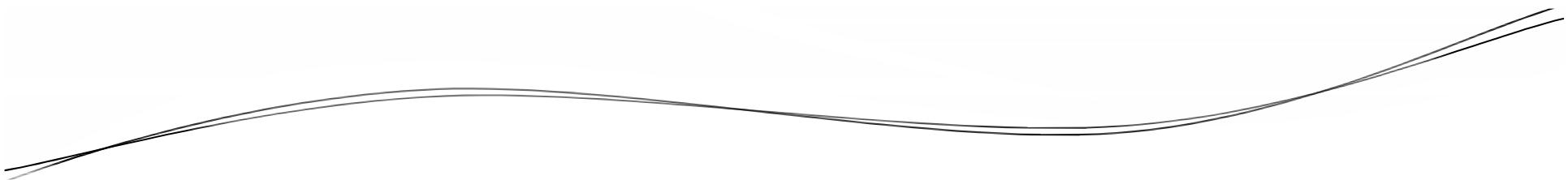
Access Modifiers Revisited

- Java provides 4 access modifiers: private, public, protected and default.
- Except private, remaining behave same unless different packages are used.



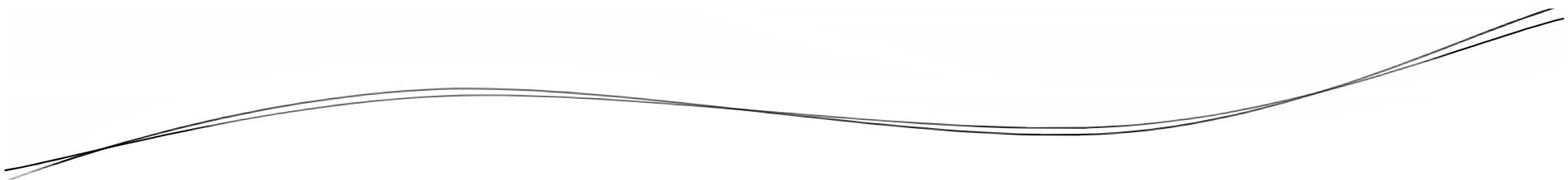
Access Modifiers Revisited

- If different packages are used then:
 - `public` makes the member accessible from anywhere.
 - `protected` makes the member accessible throughout the entire package as well as outside the package if the class is a subclass.
 - `default` makes the member accessible throughout the entire package but not outside the package. Hence it is also known as package level access modifier.



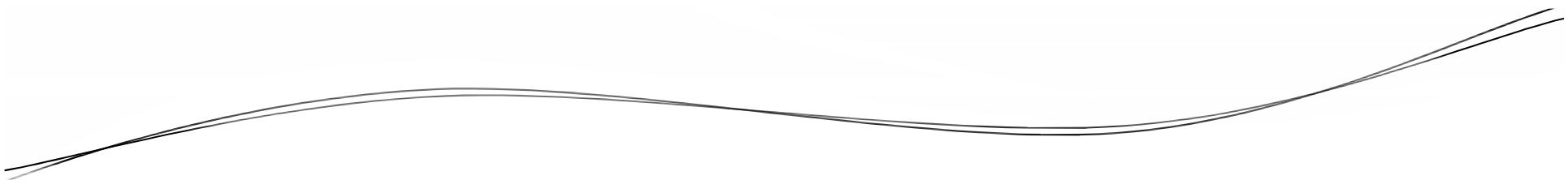
Exploring `java.lang`

By Rahul Barve



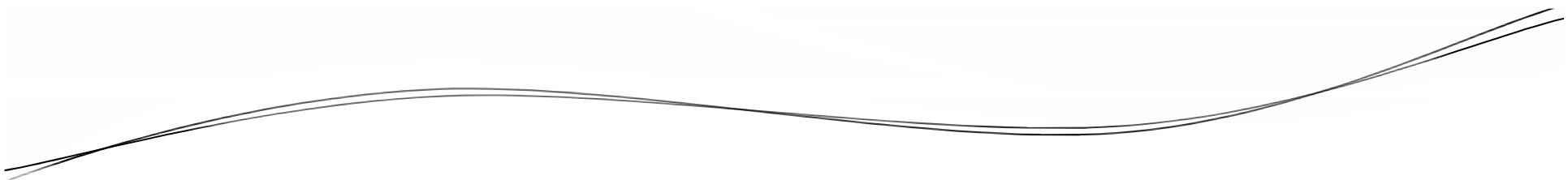
Exploring `java.lang`

- Java provides a predefined class library and all classes belong to some specific packages as it's a standard practice.
- One of the predefined packages of Java library is `java.lang`.



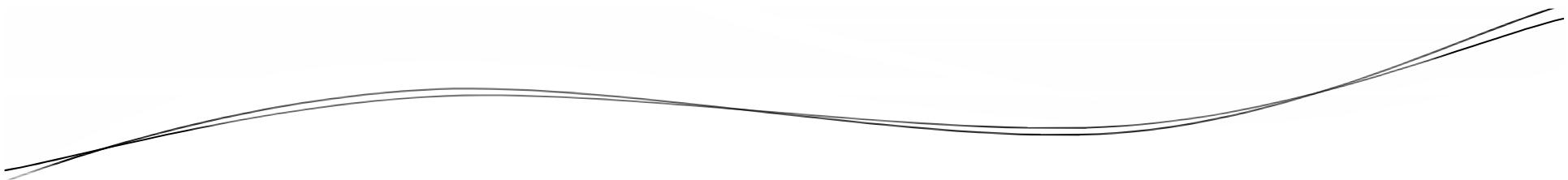
Exploring `java.lang`

- It is the package that is imported by default and hence classes belonging to `java.lang` can be used directly even without any `import` statement.



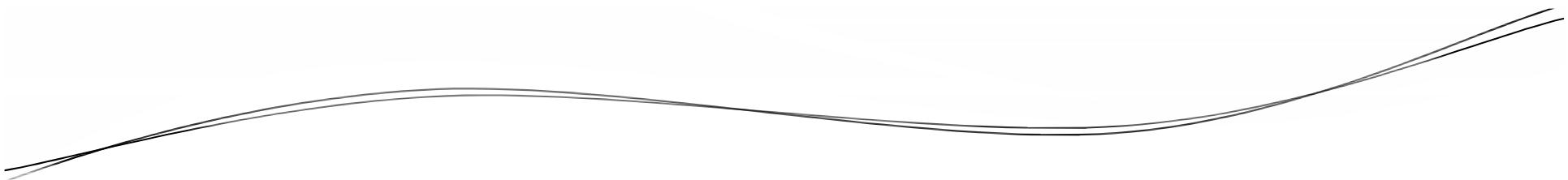
Exploring `java.lang`

- The 2 basic classes used so far:
 - `String`
 - `System`



Exploring `java.lang`

- Other Classes:
 - `Object`
 - `StringBuilder`
 - Wrapper Classes



Object Class

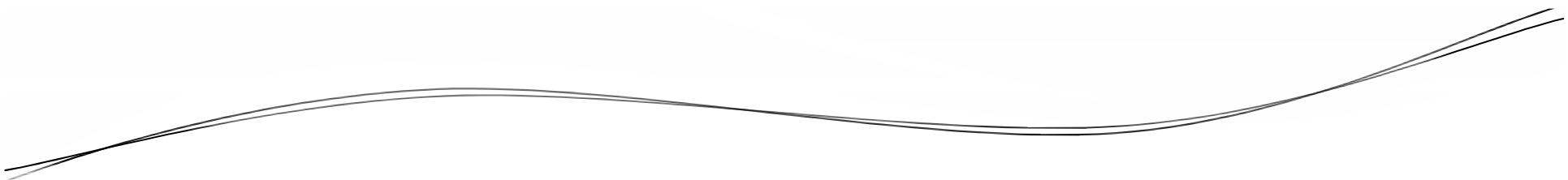
- It is the topmost class in a Java class hierarchy.
- All classes either predefined or user defined are implicitly inherited from Object.

Object Class

- A variable of type Object can be used to refer to objects of any type.
- E.g.

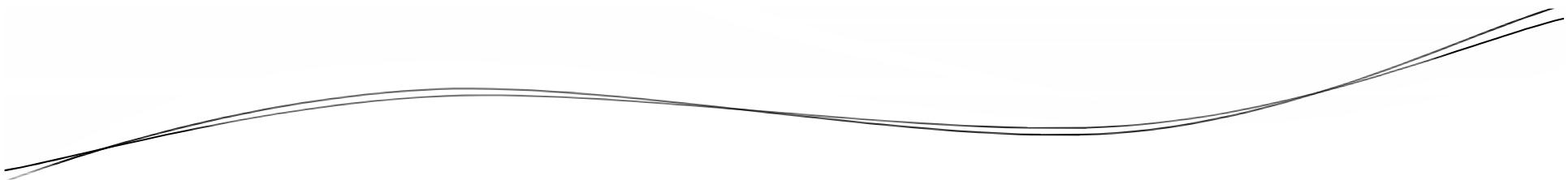
```
Object emp = new Employee();
```

```
Object cst = new Customer();
```



Methods of Object Class

- `toString()`
- `finalize()`
- `equals(Object)`
- `clone()`
- `hashCode()`
- `getClass()`
- `wait()`
- `notify()`
- `notifyAll()`

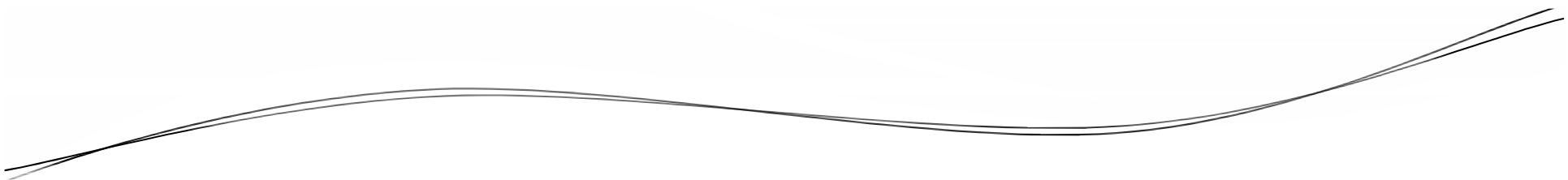


toString()

- Returns a String that represents a value of an object on which it is invoked.
- Has to be overridden in the class.
- An object can be converted into a String with explicit or implicit call.

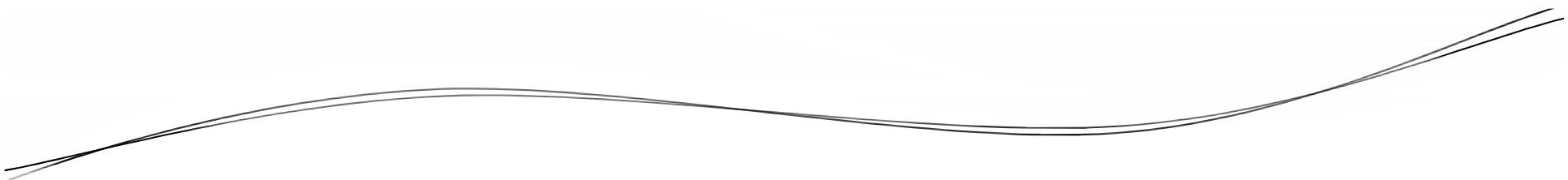
finalize()

- Can be overridden by a class; executes implicitly when GC is about to run.
- Generally overridden to perform clean-up operations.



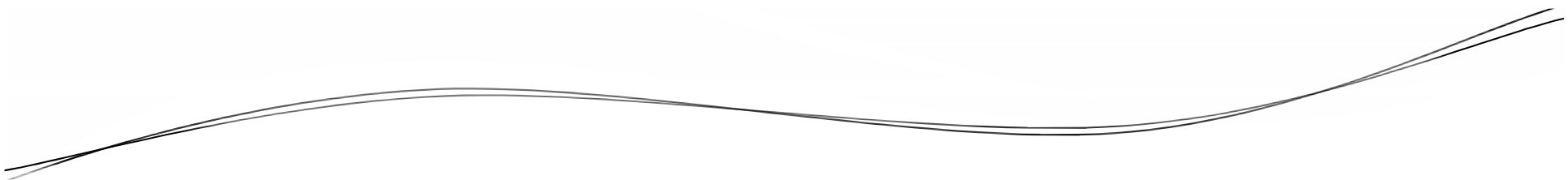
equals (Object)

- public boolean equals (Object)
- Used to test whether one object is equal to another or not.



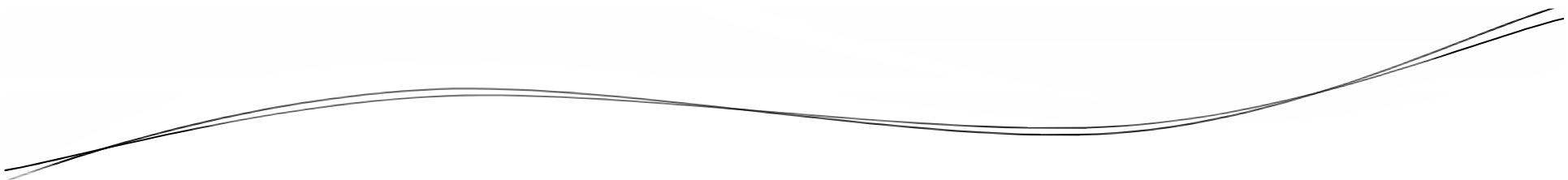
Wrapper Classes

By Rahul Barve



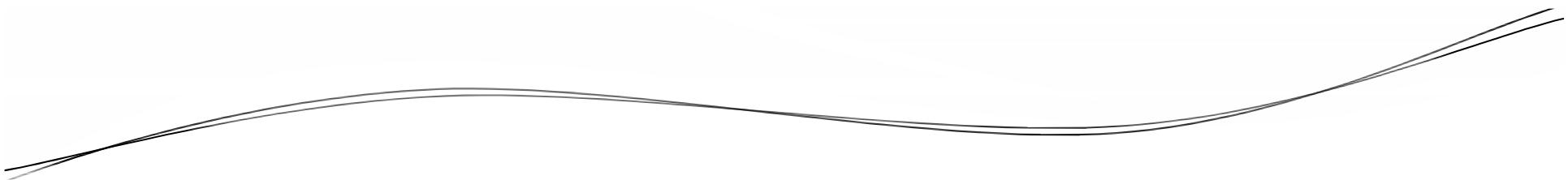
Wrapper Classes

- Java provides 8 primitive data types.
- Sometimes, there's a need to convert primitives into objects.
- All Java primitive types have class counterparts, called as wrappers or wrapper classes.



Wrapper Classes

- Byte
- Short
- Integer
- Long
- Float
- Double
- Character
- Boolean



Using Wrapper Classes

- Pre JDK 1.5

```
int val1 = 100;  
Integer iVal1 = new Integer(val1);  
int val2 = iVal1.intValue();
```

Using Wrapper Classes

- Since JDK 1.5

```
int val1 = 100;
```

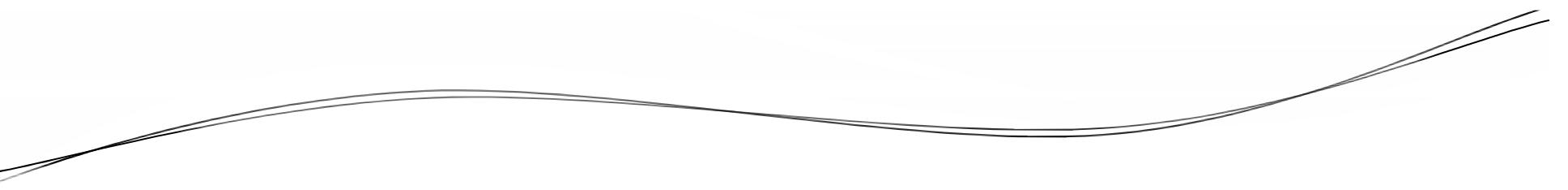
```
Integer iVal1 = val1 //Autoboxing
```

```
int val2 = iVal1; //Unboxing
```

Using Wrapper Classes

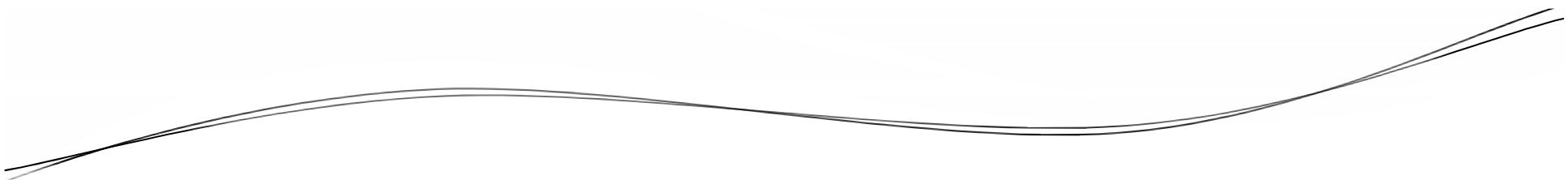
- Wrapper classes can also be used to convert a qualified String into the appropriate primitive type.
- E.g.

```
int x = Integer.parseInt("1234");  
double y =  
Double.parseDouble("3445.56");
```



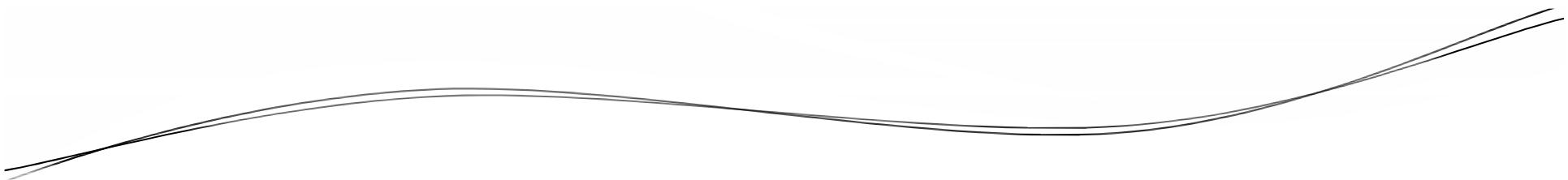
String Class

By Rahul Barve



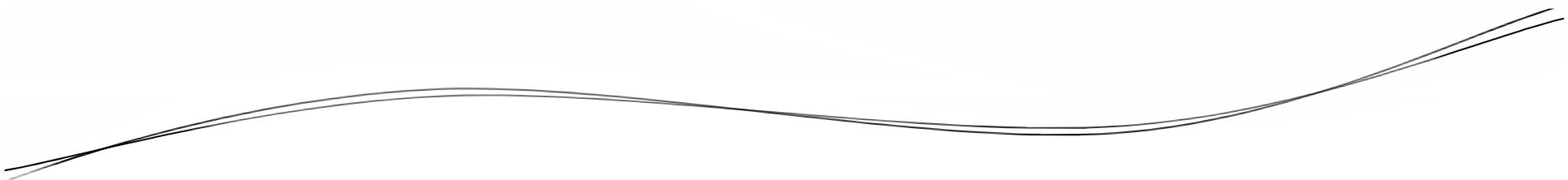
String Class

- Java library provides a predefined class String.
- Java Strings are First Class objects.
- Represents an immutable string.
- Degrades the performance.



StringBuilder Class

By Rahul Barve



StringBuilder Class

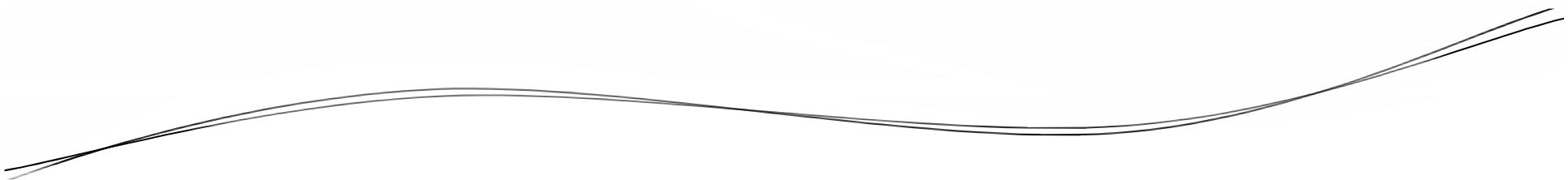
- Java library provides an alternative called as StringBuilder.
- Represents mutable strings.

StringBuilder Class

- Used to improve the performance.
- E.g.

```
StringBuilder buf;
```

```
buf = new StringBuilder("Hello");
```



Lets Summarize

- What are Packages
- Benefits of Packages
- Access Modifiers Revisited
- Classes from `java.lang`

[IceCream.java] [Sugar.java] [Fruit.java] [FruitJuice.java] [EqualsExample.java] [FinalizeExample.java] [MobilePhone.java] [ToStringExample.java] [StringBuilderWithWrapperClassExample.java] [Assignment 5.txt]

```
1 package desserts;
2
3 import food_item.*;//Imports all public classes from food_item only not from its sub-packages
4 import food_item.sweet.Sugar;
5
6 public class IceCream {
7     public static void makeMangoIceCream() {
8         Fruit mangoes = new Fruit();
9         Sugar s = new Sugar();
10    }
11
12 }
13
```

[IceCream.java] [Sugar.java] [Fruit.java] [FruitJuice.java] [EqualsExample.java] [FinalizeExample.java] [MobilePhone.java] [ToStringExample.java] [StringBuilderWithWrapperClassExample.java] [Assignment 5.txt]

```
1 package food_item.sweet;
2
3 public class Sugar {
4
5 }
6
```

[IceCream.java] [Sugar.java] [Fruit.java] [FruitJuice.java] [EqualsExample.java] [FinalizeExample.java] [MobilePhone.java] [ToStringExample.java] [StringBuilderWithWrapperClassExample.java] [Assignment 5.txt]

```
1 package food_item;
2
3 public class Fruit {
4
5     public Fruit() {
6         // TODO Auto-generated constructor stub
7     }
8
9 }
10
```

[IceCream.java] [Sugar.java] [Fruit.java] [FruitJuice.java] [EqualsExample.java] [FinalizeExample.java] [MobilePhone.java] [ToStringExample.java] [StringBuilderWithWrapperClassExample.java] [Assignment 5.txt]

```
1 package food_item;
2
3 public class FruitJuice {
4     public static void makeOrangeJuice() {
5         Fruit oranges = new Fruit();
6     }
7 }
8
9 }
```

[IceCream.java] [Sugar.java] [Fruit.java] [FruitJuice.java] EqualsExample.java [FinalizeExample.java] [MobilePhone.java] [ToStringExample.java] [StringBuilderWithWrapperClassExample.java] [Assignment 5.txt]

```
1 package object_class_demos;
2
3 public class EqualsExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         int x = 10;
8         int y = 10;
9         System.out.println("X == Y? " + (x == y));
10        MobilePhone mobile = new MobilePhone("Samsung", "Galaxy", 18500);
11        MobilePhone mobile2 = new MobilePhone("Samsung", "Galaxy", 18500);
12        //MobilePhone mobile2 = mobile;
13        System.out.println("mobile == mobile2? " + (mobile == mobile2));
14        //When '==' is used for reference variables always memory addresses are checked and not the
15        //contents (data)
16        System.out.println("mobile.equals(mobile2) ? " + (mobile.equals(mobile2)));
17        //If equals() is not overridden in the subclass, Object class equals() gets called
18        //Object class equals() works same as that of '=='
19
20    }
21
22
23 }
```

[IceCream.java] [Sugar.java] [Fruit.java] [FruitJuice.java] [EqualsExample.java] [FinalizeExample.java] [MobilePhone.java] [ToStringExample.java] [StringBuilderWithWrapperClassExample.java] [Assignment 5.txt]

```
1 package object_class_demos;
2
3 public class FinalizeExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         for(int a=1;a<=5;a++) {
8             new MobilePhone();
9         }
10        //At this stage 5 objects will be eligible for Garbage Collection
11
12        //Requesting a Garbage Collector to run
13        System.gc();
14
15    }
16
17 }
18
```

[IceCream.java] [Sugar.java] [Fruit.java] [FruitJuice.java] [EqualsExample.java] [FinalizeExample.java] [MobilePhone.java] [ToStringExample.java] [StringBuilderWithWrapperClassExample.java] [Assignment 5.txt]

```
1 package object_class_demos;
2
3 import java.util.Objects;
4
5 public class MobilePhone {
6     private String brandName;
7     private String model;
8     private int price;
9     public MobilePhone() {
10         // TODO Auto-generated constructor stub
11     }
12     public MobilePhone(String brandName, String model, int price) {
13         this.brandName = brandName;
14         this.model = model;
15         this.price = price;
16     }
17     public String getBrandName() {
18         return brandName;
19     }
20     public void setBrandName(String brandName) {
21         this.brandName = brandName;
22     }
23     public String getModel() {
24         return model;
25     }
26     public void setModel(String model) {
27         this.model = model;
28     }
29     public int getPrice() {
30         return price;
```

[IceCream.java] [Sugar.java] [Fruit.java] [FruitJuice.java] [EqualsExample.java] [FinalizeExample.java] [MobilePhone.java] [ToStringExample.java] [StringBuilderWithWrapperClassExample.java] [Assignment 5.txt]

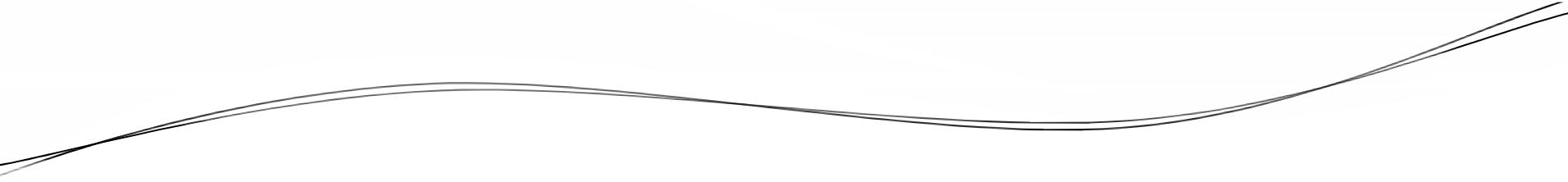
```
31     }
32     public void setPrice(int price) {
33         this.price = price;
34     }
35     @Override
36     public void finalize() {
37         System.out.println("Object is getting destroyed...");
38     }
39     @Override
40     public String toString() {
41         return "MobilePhone [brandName=" + brandName + ", model=" + model + ", price=" + price + "]";
42     }
43     @Override
44     public int hashCode() {
45         return Objects.hash(brandName, model, price);
46     }
47     @Override
48     public boolean equals(Object obj) {
49         if (this == obj)
50             return true;
51         if (obj == null)
52             return false;
53         if (getClass() != obj.getClass())
54             return false;
55         MobilePhone other = (MobilePhone) obj;
56         return Objects.equals(brandName, other.brandName) && Objects.equals(model, other.model) && price == other.price;
57     }
58 }
59 }
60 }
```

```
IceCream.java x Sugar.java x Fruit.java x FruitJuice.java x EqualsExample.java x FinalizeExample.java x MobilePhone.java x ToStringExample.java x StringBuilderWithWrapperClassExample.java x Assignment 5.txt x
1 package object_class_demos;
2
3 public class ToStringExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         int x = 100;
8         System.out.println(x);
9
10        MobilePhone mobile = new MobilePhone("Samsung", "Galaxy", 18500);
11        System.out.println(mobile);
12        //In the above statement, JRE identifies dynamic type of 'mobile' ---> MobilePhone
13        //And checks for toString() method in the respective class. If found, invokes that
14        //Otherwise invokes the method from its super class ----> Object
15
16        //At line no 11, the call to toString() is implicit
17
18        String mobileData = mobile.toString(); //This is an explicit call
19        System.out.println(mobileData);
20
21        //In Java, there are some methods which get called implicitly by JRE. Such methods
22        //are known as Callback methods e.g toString()
23
24    }
25
26 }
27
28
29
30
```

[IceCream.java] [Sugar.java] [Fruit.java] [FruitJuice.java] [EqualsExample.java] [FinalizeExample.java] [MobilePhone.java] [ToStringExample.java] [StringBuilderWithWrapperClassExample.java] [Assignment 5.txt]

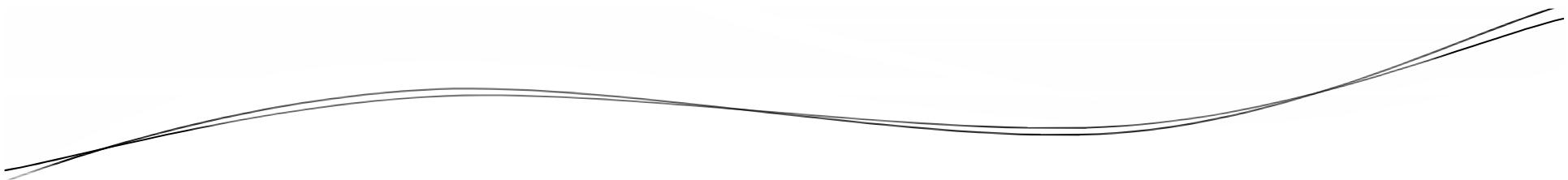
```
1 package wapper_class_string_builder;
2
3 public class StringBuilderWithWrapperClassExample {
4
5     public static void main(String[] args) {
6         // This program accepts 3 command line arguments: person name, age, weight
7         String name = args[0];
8         int age = Integer.parseInt(args[1]);
9         float weight = Float.parseFloat(args[2]);
10
11        //Creating an empty StringBuilder
12        StringBuilder builder = new StringBuilder();
13        //Start adding data into StringBuilder using overloaded append() methods
14        builder.append("Hello ");
15        builder.append(name);
16        builder.append(". Your age is ");
17        builder.append(age);
18        builder.append(" years and your weight is ");
19        builder.append(weight);
20        builder.append(" Kgs");
21
22        System.out.println(builder);
23
24    }
25
26
27 }
```

```
1 Time Limit: 30 Minutes
2
3 Create a class Shirt with following attributes:
4     String brandName (E.g. Zodiac, Peter England, Arrow etc.)
5     String color
6     String type (E.g. Checks, Lining, Plain etc)
7     int price
8
9 Create another class ShirtCollection with following attribute and methods:
10    public static Shirt shirts[]
11    public static void setShirts(Shirt [] shirts)
12    public static void showAllShirts(String type)
13    public static void
14        showAllShirts(int minPrice, int maxPrice)
15    public static boolean isShirtAvailable(Shirt shirt)
16
17
18
19 Write a main program that accepts a command line argument for
20 the type of the shirt and displays all the data about the shirts
21 that match with that type.
22
23 Write another main program that accepts command line arguments
24 for a price range (min limit and max limit) of the shirt
25 and display all the data about the shirts having prices within
26 that range
27
28 Write one more main program that accepts command line arguments
29 for all the shirt details:
30     brandName, color, type, price
31 and indicates whether the shirt is available or not
32
```



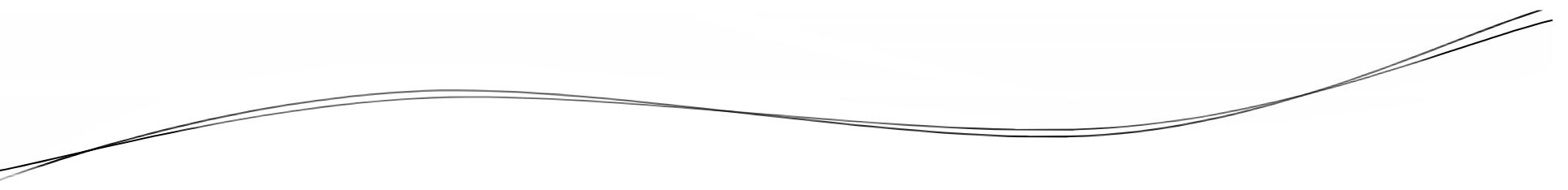
Interfaces

By Rahul Barve



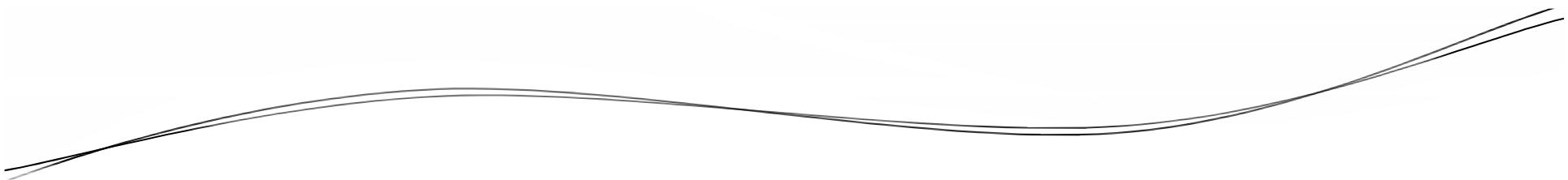
Objectives

- Introduction to Interfaces
- Why Interfaces
- Interface Example
- Understand Interface Rules.
- Abstract Class Vs. Interface
- New Features



Interface

By Rahul Barve



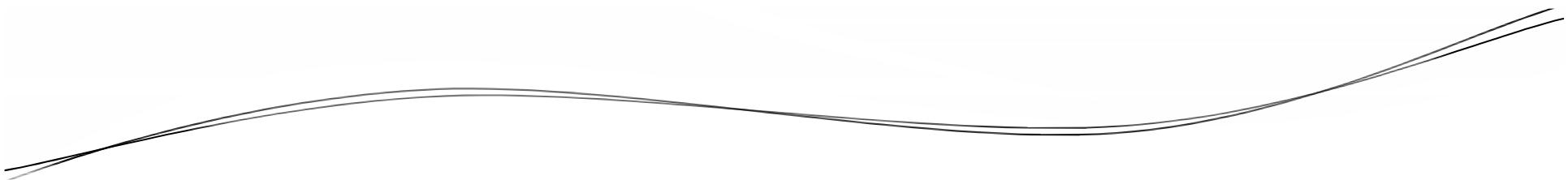
Interface

- Interface is a collection of abstract methods and possibly final variables.
- Used to declare methods where implementation is not available.

Interface

- An interface is declared using a keyword interface.
- E.g.

```
public interface MyInterface{  
    int myVar = 100;  
    void myMethod();  
}
```



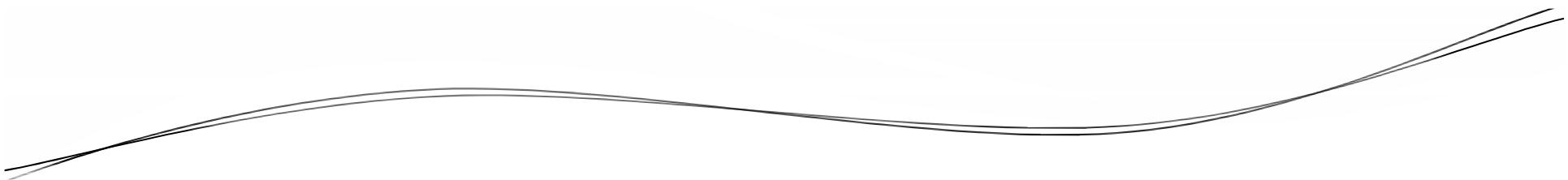
Interface

- Once an interface is created, it can be further used by creating an implementation class for that interface.

Interface

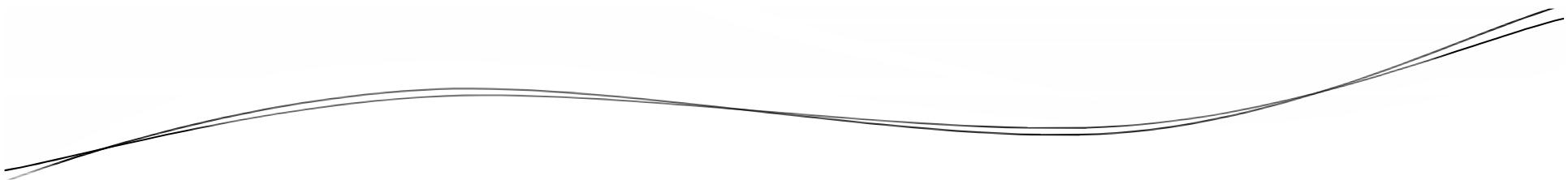
- E.g.

```
public class MyClass  
    implements MyInterface {  
    public void myMethod() {  
        System.out.println(myVar);  
    }  
}
```



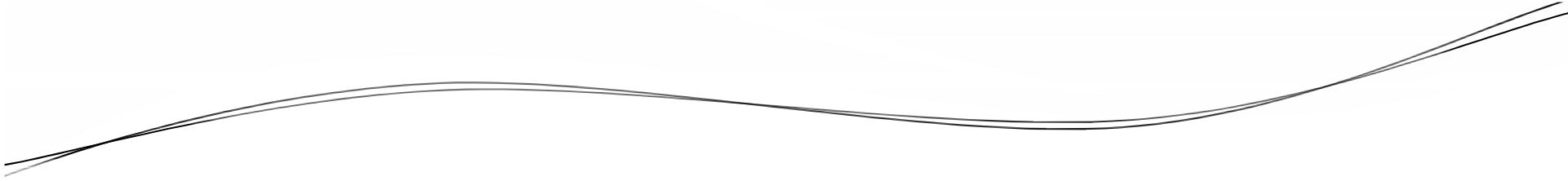
Why Interface

By Rahul Barve



Why Interface

- An interface is used to expand the scope of polymorphism.
- Achieving multiple inheritance in the context of methods.
- Used for loose coupling.

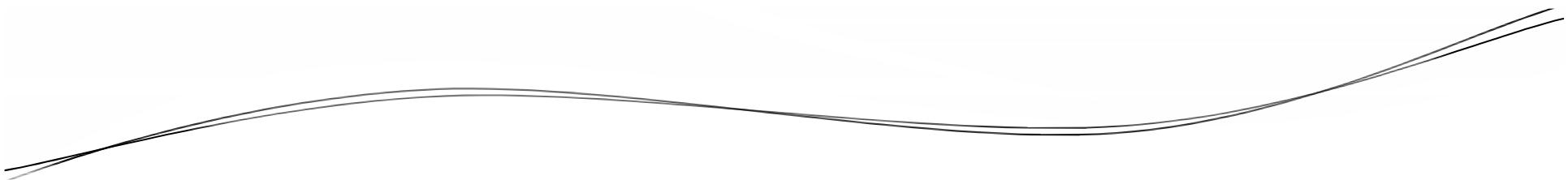


Interface Rules

- Possible associations:
 - Class extends Class
 - Class implements Interface(s)
 - Interface extends Interface(s)

Interface Rules

- Methods of interface are by default public and abstract
- Variables of interface are by default public, static and final .
- A class that implements interface, must implement all the methods of that interface; otherwise must be declared abstract.

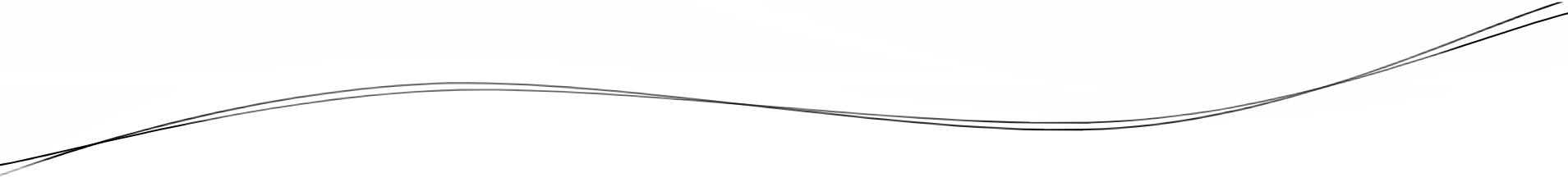


Interface Rules

- An object of a class is always compatible with the interface type.
- An interface type is always compatible with Object.

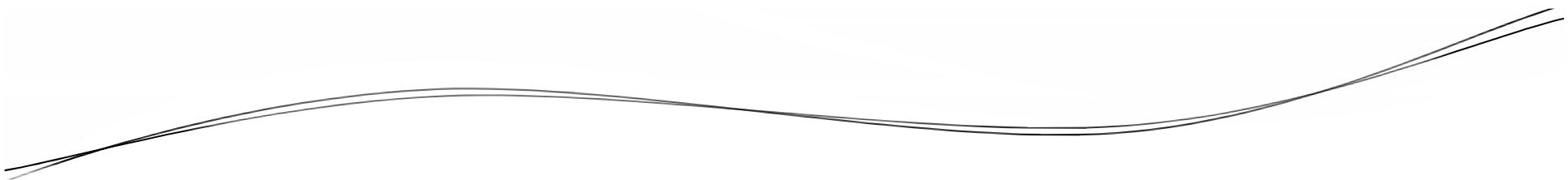
Abstract Class Vs. Interface

- A class can extend only one abstract class.
- Useful to achieve polymorphism when classes are co-related.
- Can contain concrete methods also.
- A class can implement any no of interfaces.
- Useful to achieve polymorphism when classes are not co-related.
- Generally contains only abstract methods.



New Features

By Rahul Barve



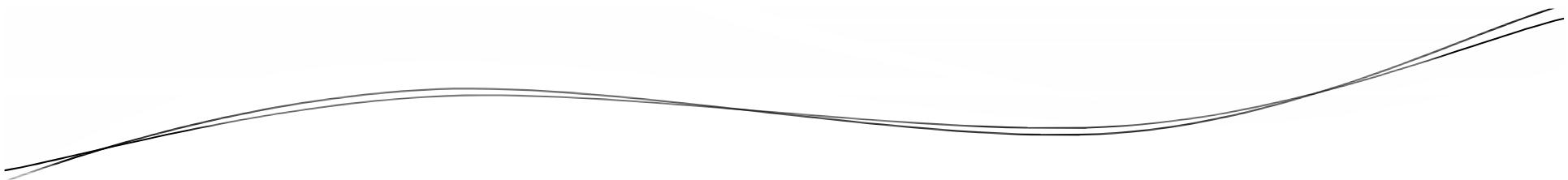
New Features

- Since JDK 1.8, it is possible to define methods within an interface provided they are declared as either default or static.
- This feature enables to add a new functionality in the interfaces without breaking the existing contract of the implementing classes.

Default and Static Methods

- E.g.

```
public interface MyInterface {  
    default void m1() {  
        //Some Code  
    }  
    static void m2() {  
    }  
}
```



Lets Summarize

- What are Interfaces
- Why Interfaces
- Interface Examples
- Interface Rules.
- Abstract Class Vs. Interface
- New Features

DollarToRupeeConverter.java ✘ RupeeToPoundConverter.java ✘ InterfaceExample.java ✘ ForexOperation.java ✘ InterfaceLooseCouplingExample.java ✘ Printable.java ✘ PrintableImpl.java ✘ DefaultAndStaticMethodExample.java ✘

```
1 package interfaces;
2
3 public class DollarToRupeeConverter implements CurrencyConverter {
4
5     @Override
6     public float doConvert(float amountInDollars) {
7         // TODO Auto-generated method stub
8         float amountInRupees = amountInDollars * DOLLAR_TO_RUPEE;
9         return amountInRupees;
10    }
11
12 }
13
```

DollarToRupeeConverter.java x RupeeToPoundConverter.java x InterfaceExample.java x ForexOperation.java x InterfaceLooseCouplingExample.java x Printable.java x PrintableImpl.java x DefaultAndStaticMethodExample.java x

```
1 package interfaces;
2
3 public class RupeeToPoundConverter implements CurrencyConverter {
4
5     @Override
6     public float doConvert(float amountInRupees) {
7         // TODO Auto-generated method stub
8         float amountInPounds = amountInRupees / POUND_TO_RUPEE;
9         return amountInPounds;
10    }
11
12 }
13
```

```
1 package interfaces;
2
3 public class InterfaceExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         System.out.println("Welcome to Foreign Exchange....");
8         System.out.println("Before we begin, let's have a look at conversion rates: ");
9         System.out.println("Dollar to Rupees: " + CurrencyConverter.DOLLAR_TO_RUPEE);
10        System.out.println("Pound to Rupees: " + CurrencyConverter.POUND_TO_RUPEE);
11        //trying to modify dollar rate:
12        //CurrencyConverter.DOLLAR_TO_RUPEE = 83.45f;
13        //The above statement is invalid because interface variables are by default final
14        CurrencyConverter forexConverter;
15        forexConverter = new DollarToRupeeConverter();
16        float usDollars = 10000;
17        float indianRupees = forexConverter.doConvert(usDollars);
18        System.out.println("Amount in INR for USD: " + usDollars + " is " + indianRupees);
19        System.out.println("-----");
20        forexConverter = new RupeeToPoundConverter();
21        indianRupees = 5000000;
22        float gbPounds = forexConverter.doConvert(indianRupees);
23        System.out.println("Amount in GBP for INR: " + indianRupees + " is " + gbPounds);
24
25    }
26
27
28
29
30
31
32 }
```

DollarToRupeeConverter.java X RupeeToPoundConverter.java X InterfaceExample.java X ForexOperation.java X InterfaceLooseCouplingExample.java X Printable.java X PrintableImpl.java X DefaultAndStaticMethodExample.java X

```
1 package interfaces;
2
3 public class ForexOperation {
4     private float amountInput;
5     private CurrencyConverter converter;
6     public ForexOperation() {
7         amountInput = 10000;
8         //By default DollarToRupee conversion is used
9         converter = new DollarToRupeeConverter();
10    }
11    public ForexOperation(float amountInput, CurrencyConverter converter) {
12        this.amountInput = amountInput;
13        this.converter = converter;
14    }
15
16    public float getConvertedAmount() {
17        float amountOutput = converter.doConvert(amountInput);
18        return amountOutput;
19    }
20    public float getAmountInput() {
21        return amountInput;
22    }
23    public void setAmountInput(float amountInput) {
24        this.amountInput = amountInput;
25    }
26    public CurrencyConverter getConverter() {
27        return converter;
28    }
29    public void setConverter(CurrencyConverter converter) {
30        this.converter = converter;
31    }
32 }
```

DollarToRupeeConverter.java | RupeeToPoundConverter.java | InterfaceExample.java | ForexOperation.java | InterfaceLooseCouplingExample.java | Printable.java | PrintableImpl.java | DefaultAndStaticMethodExample.java

```
1 package interfaces;
2
3 public class InterfaceLooseCouplingExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7
8         //Using the default amount and the converter
9         ForexOperation forex = new ForexOperation();
10        float convertedAmount = forex.getConvertedAmount();
11        System.out.println(convertedAmount);
12
13        //Changing the amount as well as the converter
14        forex.setAmountInput(500000);
15        forex.setConverter(new RupeeToPoundConverter());
16        convertedAmount = forex.getConvertedAmount();
17        System.out.println(convertedAmount);
18
19    }
20
21 }
22 }
```

DollarToRupeeConverter.java RupeeToPoundConverter.java InterfaceExample.java ForexOperation.java InterfaceLooseCouplingExample.java Printable.java PrintableImpl.java DefaultAndStaticMethodExample.java

```
1 package interfaces;
2
3 public interface Printable {
4     void print(); //Abstract method
5     default void startPrinting() { //Default implemented method
6         System.out.println("Printing starts...");
7     }
8     static void help() { //Static implemented method
9         System.out.println("This interface is meant for printing the document");
10    }
11 }
12
13 }
```

DollarToRupeeConverter.java RupeeToPoundConverter.java InterfaceExample.java ForexOperation.java interfaceLooseCouplingExample.java Printable.java PrintableImpl.java DefaultAndStaticMethodExample.java

```
1 package interfaces;
2
3 public class PrintableImpl implements Printable {
4
5     @Override
6     public void print() {
7         // TODO Auto-generated method stub
8         //Start the printing
9         startPrinting(); //Invoking the default method
10        System.out.println("Printing in progress");
11    }
12
13
14 }
15
```

DollarToRupeeConverter.java RupeeToPoundConverter.java InterfaceExample.java ForexOperation.java InterfaceLooseCouplingExample.java Printable.java PrintableImpl.java DefaultAndStaticMethodExample.java

```
1 package interfaces;
2
3 public class DefaultAndStaticMethodExample {
4
5     public static void main(String[] args) {
6         Printable.help(); //Invokes static method
7         Printable p = new PrintableImpl();
8         p.print();
9
10    }
11
12 }
13
```

Assignment 6 - Notepad
File Edit Format View Help
Time Limit: 45 Minutes

Create a class Department with following attributes:

 name (String)
 location (String)

Create a class Employee with following attributes:

 empNo (int)
 name (String)
 salary (float)
 department (Department)

Create an interface EmployeeProcessor with following

method declaration:

 float processEmployees(Employee [], String);

Create 2 implementation classes for the interface

EmployeeProcessor as per the following:

1. Class names will be of your choice
2. First implementation class must return the sum of salaries
 of the employees as per the department name
3. Second implementation class must return the number of
 employees as per the department location

Create a class EmployeeOperation with following attributes:

 allEmployees (Employee [])
 operationResult (float)
 processor (EmployeeProcessor)

Implement a method:

public void perform() in such a way that it must perform
the operation using the 'processor' and store the result
into the permanent field 'operationResult'.

Write a main program to print the result of EmployeeOperation

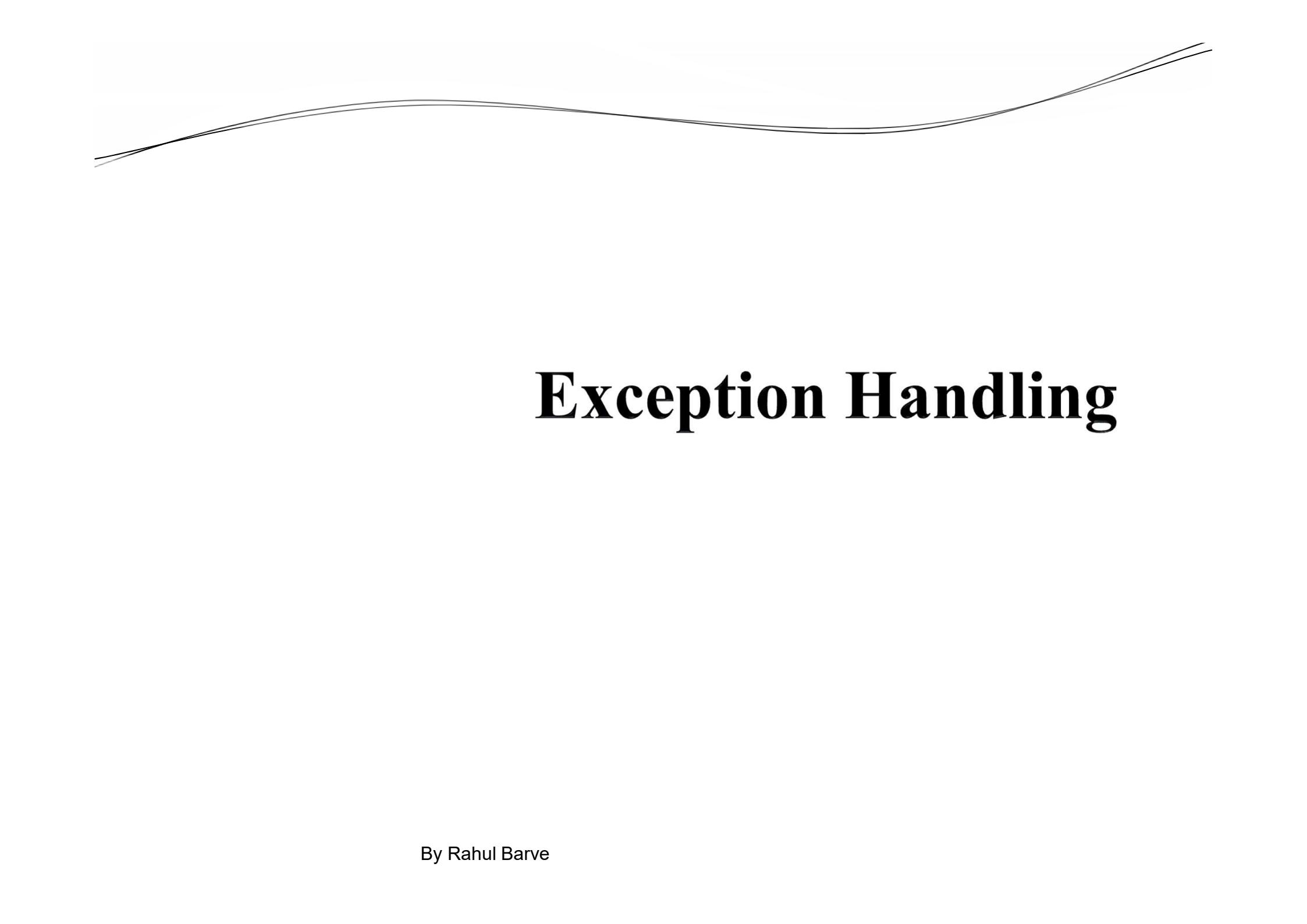
Department
name (String)
location (String)

Employee
empNo (int)
name (String)
salary (float)
department (Department)

EmployeeProcessor (I)
processEmployees
(Employee [], String);

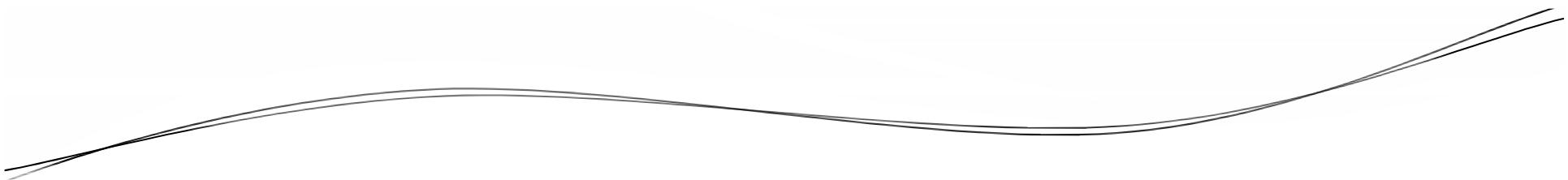
IMPL 1 and IMPL 2

EmployeeOperation
allEmployees (Employee [])
operationResult (float)
processor
(EmployeeProcessor)



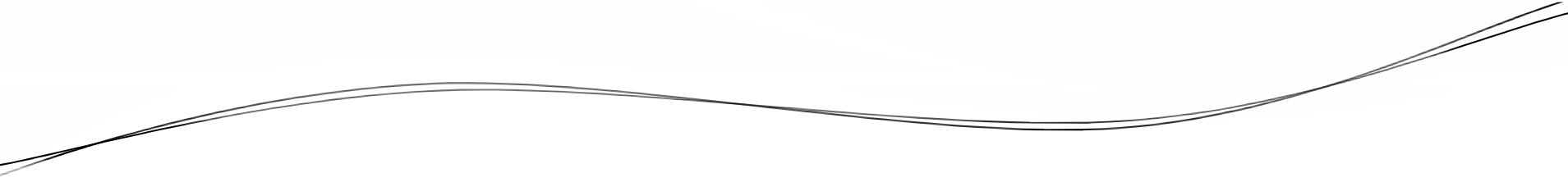
Exception Handling

By Rahul Barve



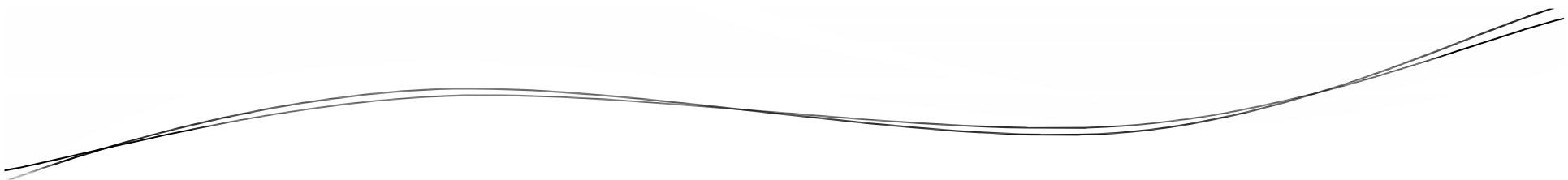
Objectives

- Introduction to Exception Handling
- Need for Exception Handling
- A Simple Example
- Understand Exception Hierarchy
- Exception Types
- Exception Handling Keywords
- User Defined Exceptions



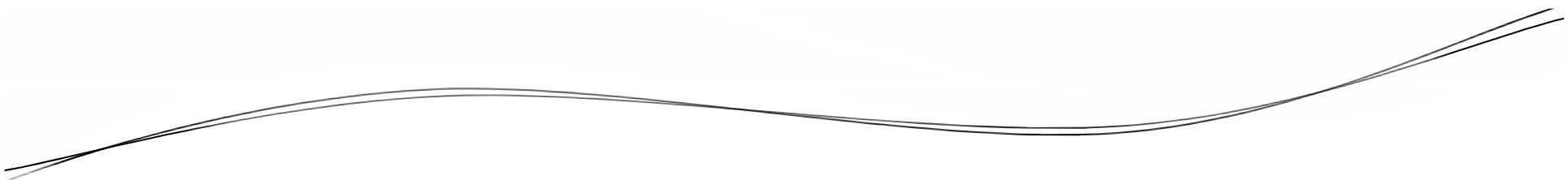
Exception Handling

By Rahul Barve



Exception Handling

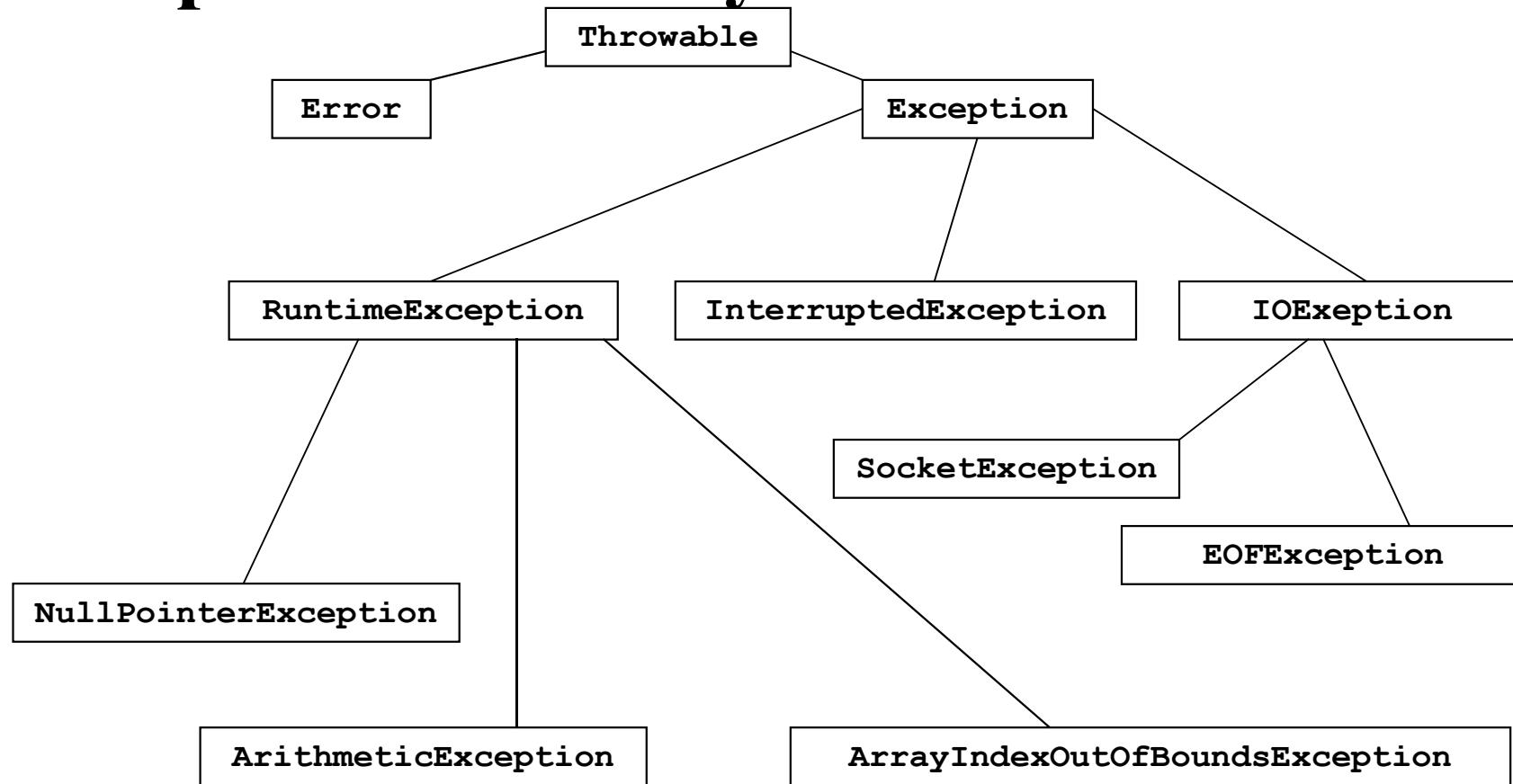
- Exception Handling is an object oriented way of handling errors which occur during program's execution.
- Problem solving code is decoupled from error handling code and hence the program is less complex.



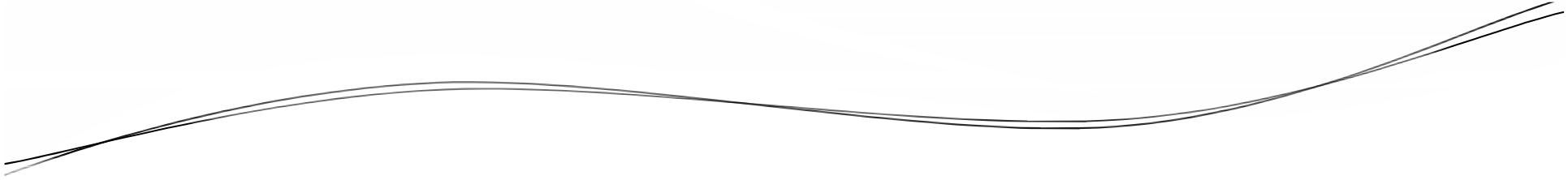
Exception Handling

- Exceptions in Java are actual objects.
- Exception objects encapsulate the error information.
- Exceptions are created when an abnormal situation occurs in a Java program.

Exception Hierarchy

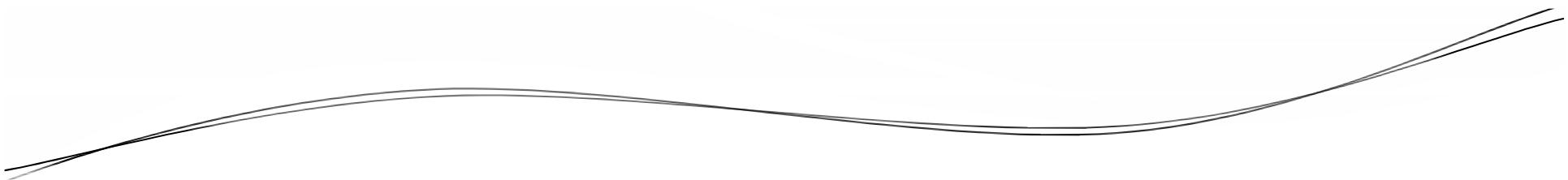


By Rahul Barve



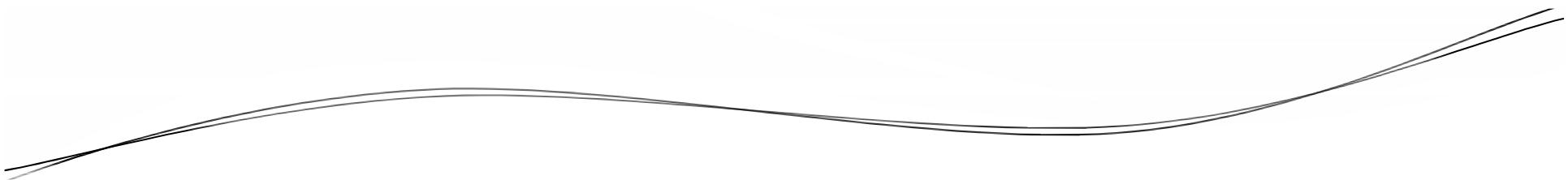
Exception Hierarchy

- The topmost class in the exception hierarchy is Throwable.
- It belongs to java.lang package.
- It includes all types of runtime errors and hence it is derived by Error and Exception.



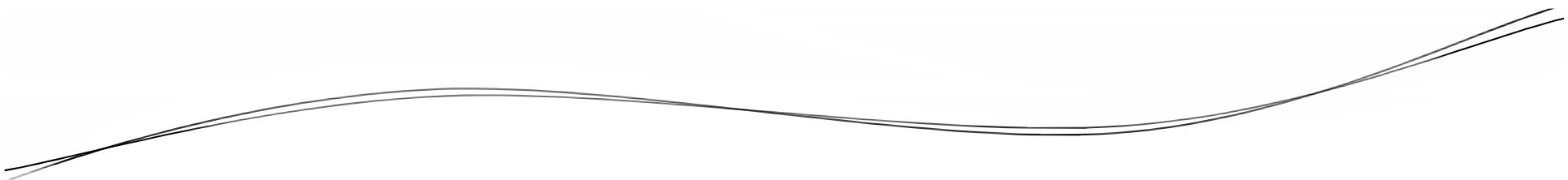
Error

By Rahul Barve



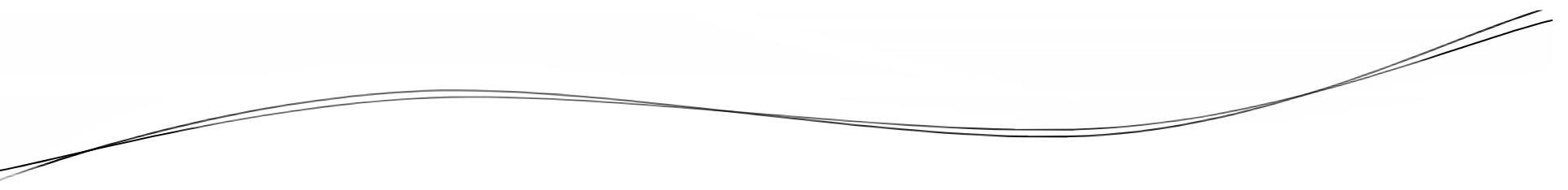
Error

- Error indicates a runtime error which is not under the control of a developer.
- It describes resource exhaustion in JVM.



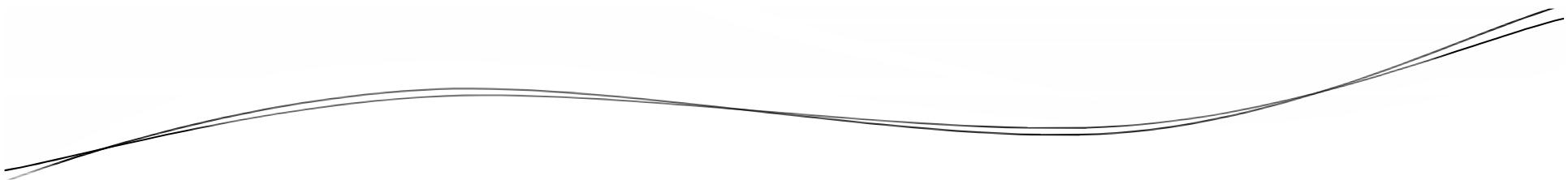
Error

- Rare and usually fatal.
- E.g.
 - StackOverflowError
 - OutOfMemoryError



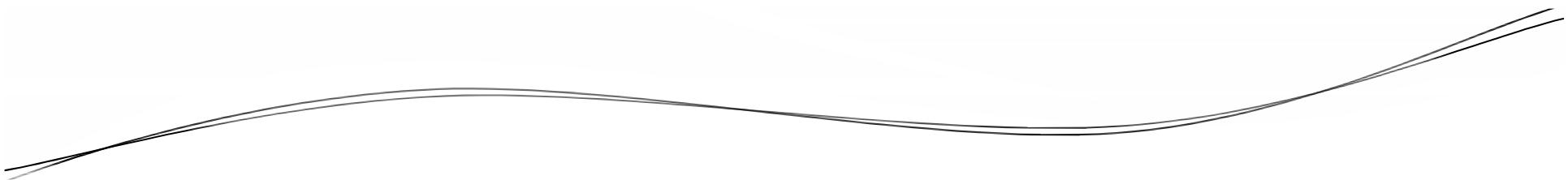
Exception

By Rahul Barve



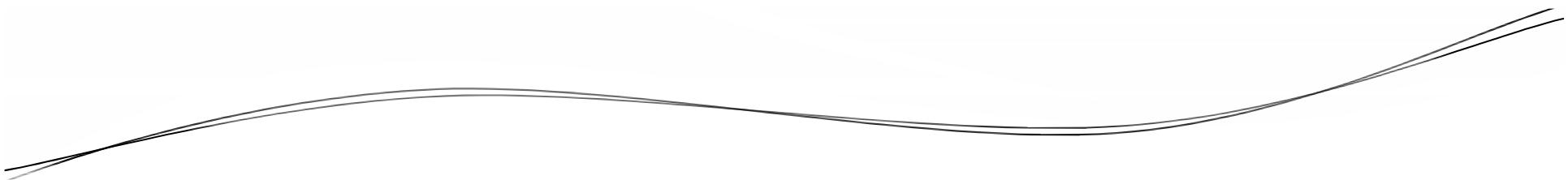
Exception

- Exception indicates a runtime error which is under the control of a developer.
- Frequent but not fatal.



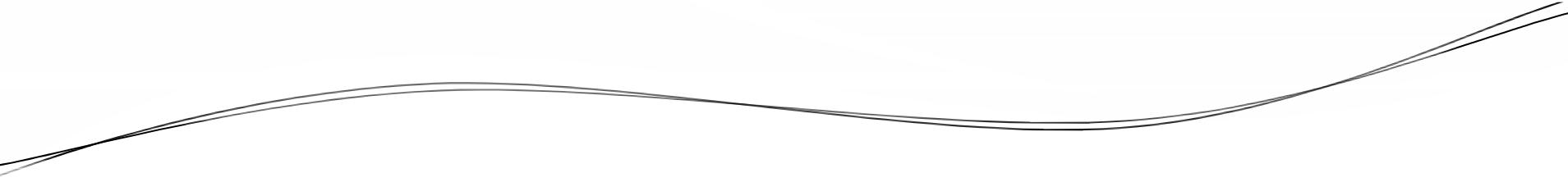
Exception Types

By Rahul Barve



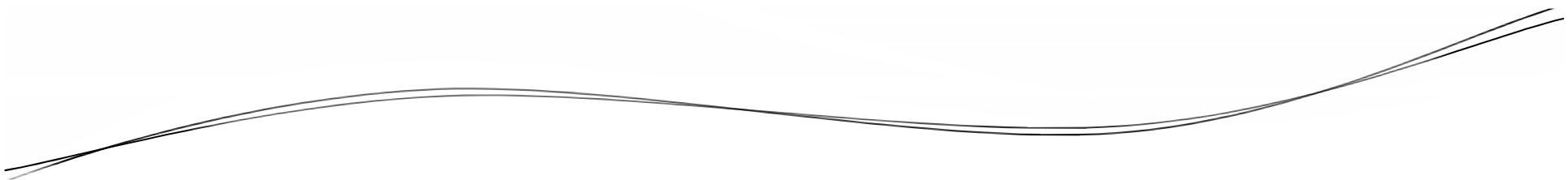
Exception Types

- Exceptions are further divided into 2 types:
 - Unchecked Exceptions
 - Checked Exceptions



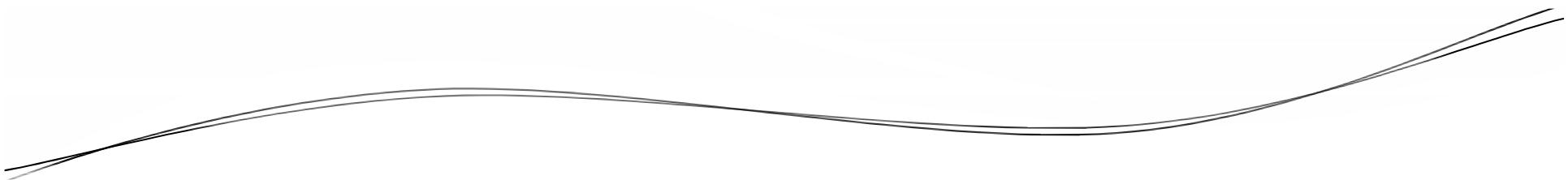
Unchecked Exceptions

By Rahul Barve



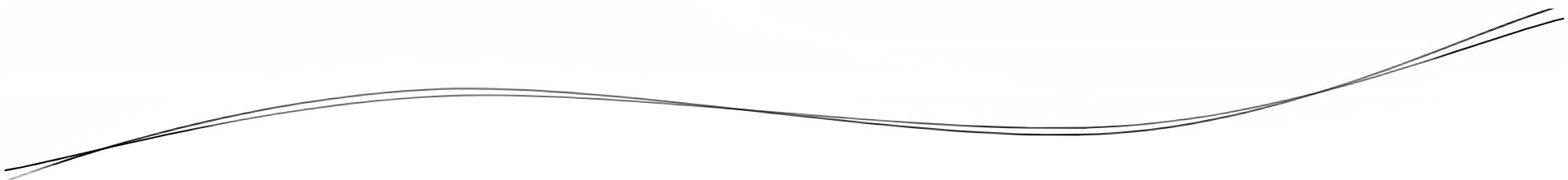
Unchecked Exceptions

- Unchecked Exceptions occur due to programming mistakes i.e. a non robust code.
- They are also called as Runtime Exceptions and hence expressed using a class `RuntimeException`.
- All classes descended from `RuntimeException` are runtime exceptions.



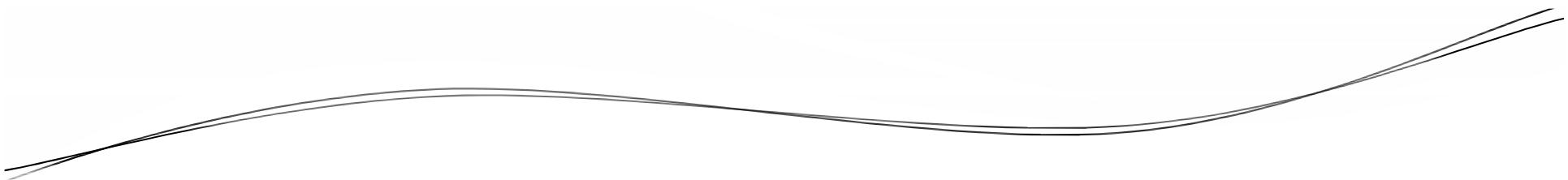
Unchecked Exceptions

- Include problems such as:
 - Bad cast
 - Out of bounds array access
 - A null pointer access



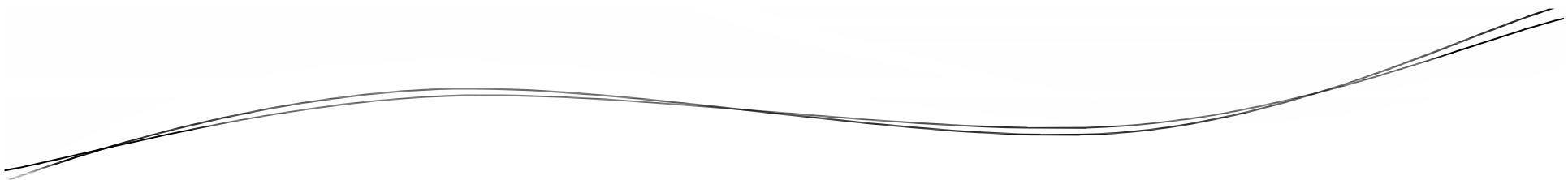
Unchecked Exceptions

- NullPointerException
- ArrayIndexOutOfBoundsException
- ArithmeticException



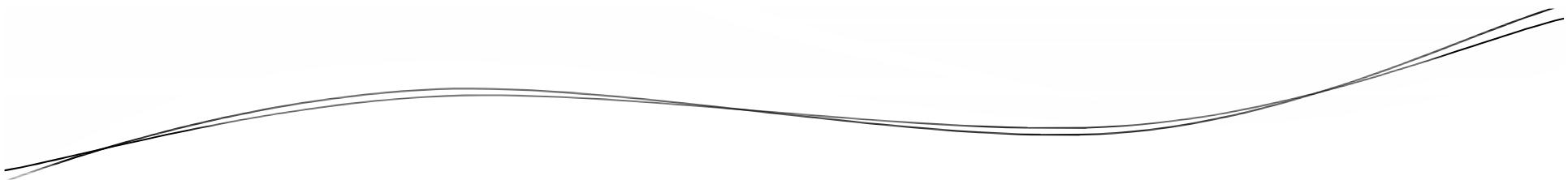
Checked Exceptions

By Rahul Barve



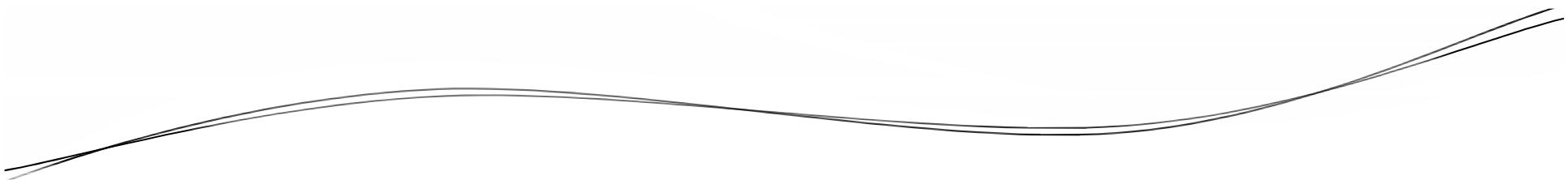
Checked Exceptions

- Occur due to problems in the environment settings.
- These exceptions are enforced by a compiler to be handled.
- These exceptions are expressed with the help of classes which are not descendants of `RuntimeException`.



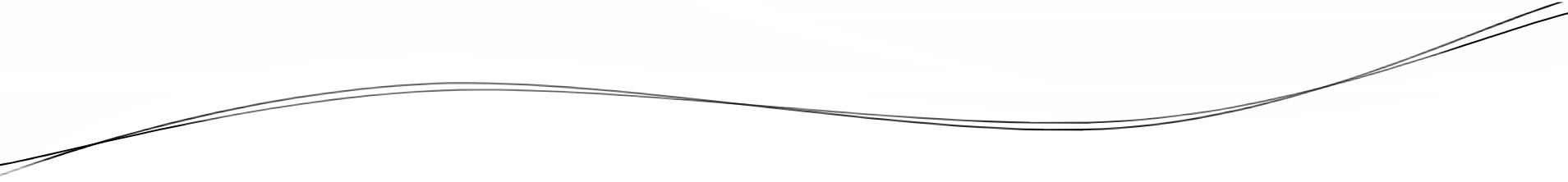
Checked Exceptions

- Problems include such as:
 - Opening a file that does not exist.
 - Unable to load a class.



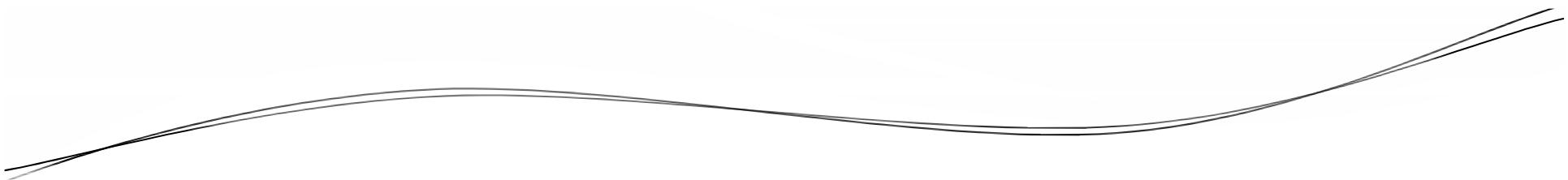
Checked Exceptions

- FileNotFoundException
- ClassNotFoundException



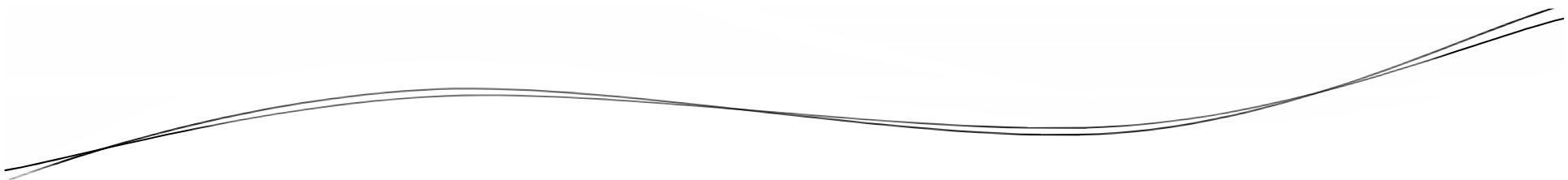
Handling Exceptions

By Rahul Barve



Handling Exceptions

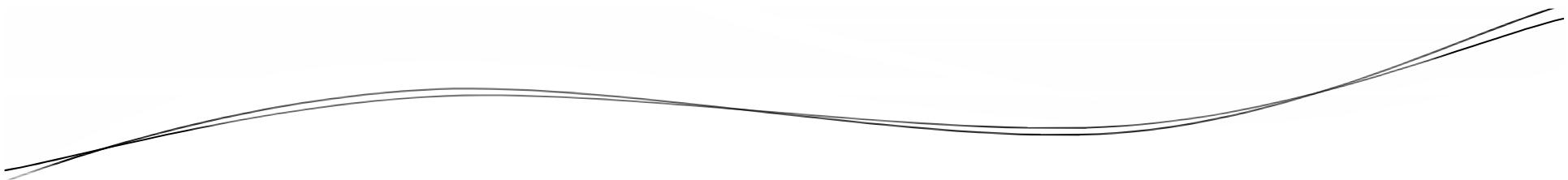
- To handle the exceptions, it is necessary to enclose the statements, which are probable to fire an exception, within a `try` block.



Handling Exceptions

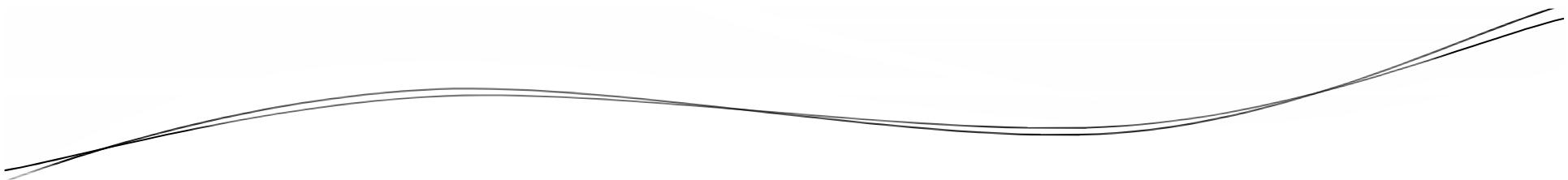
- E.g.

```
try {  
    //Statement 1  
    //Statement 2  
}
```



Handling Exceptions

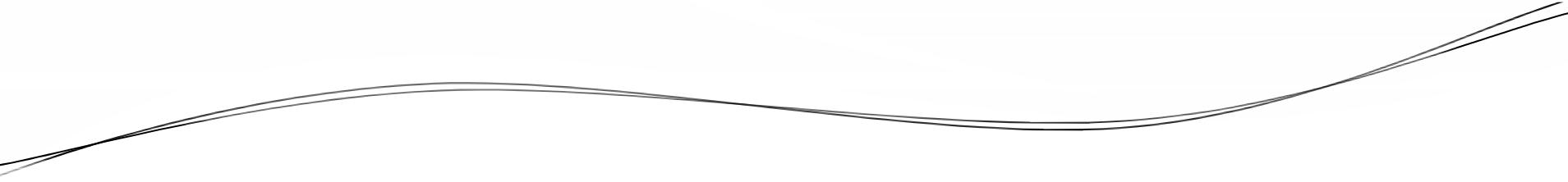
- If an exception is raised, it needs to be handled using an exception handler.
- This is done using a catch block.



Handling Exceptions

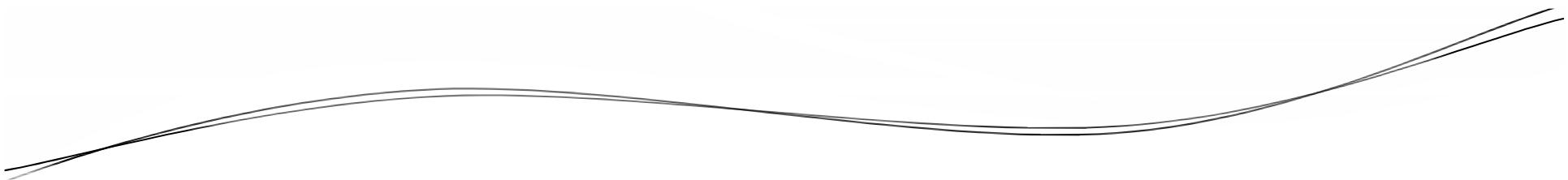
- E.g.

```
catch (<Exception Type> <ref-name>) {  
    //Statements  
}
```



Handling Multiple Exceptions

By Rahul Barve



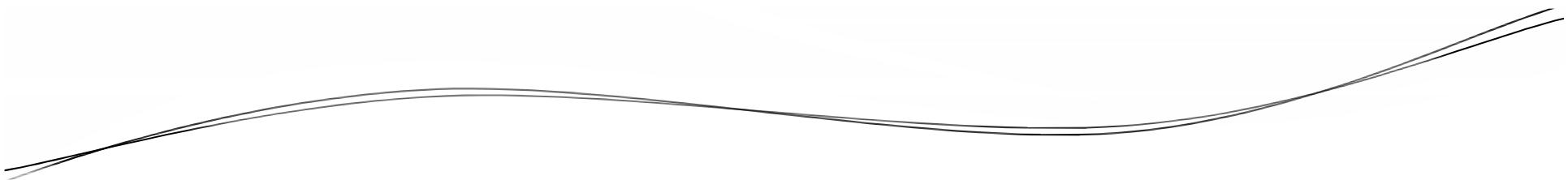
Handling Multiple Exceptions

- If a block of code is capable for firing multiple exceptions, it is possible to handle them by providing multiple catch blocks.

Handling Multiple Exceptions

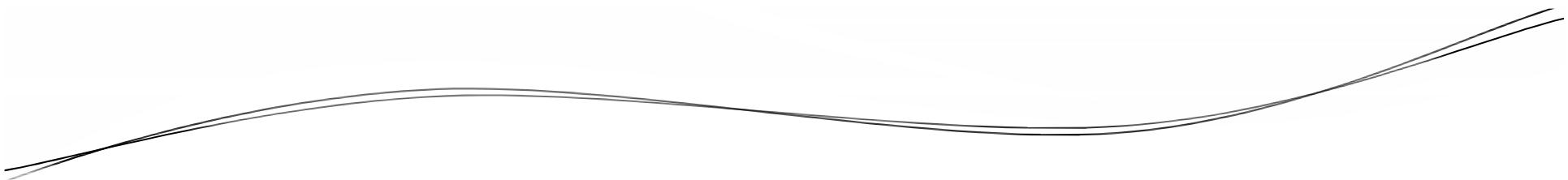
- E.g.

```
try {  
    //Statements  
}  
catch(<exception type> <ref-name>) {  
    //Statements  
}  
catch(<exception type> <ref-name>) {  
    //Statements  
}
```



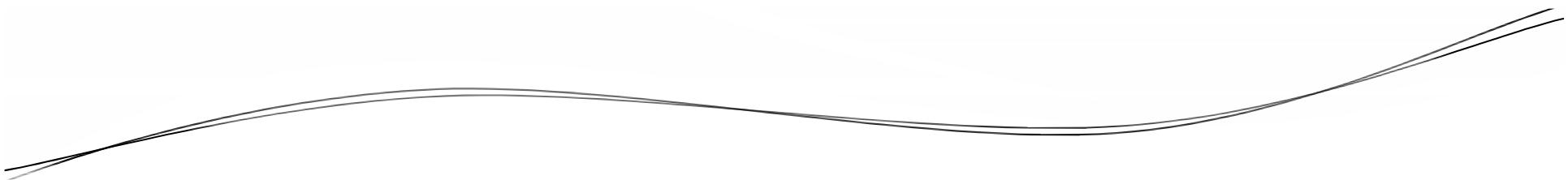
Handling Multiple Exceptions

- When using multiple catch blocks, if the exception types represent parent-child relationship, then the catch block of sub type must appear before the catch block of super type.



Handling Multiple Exceptions

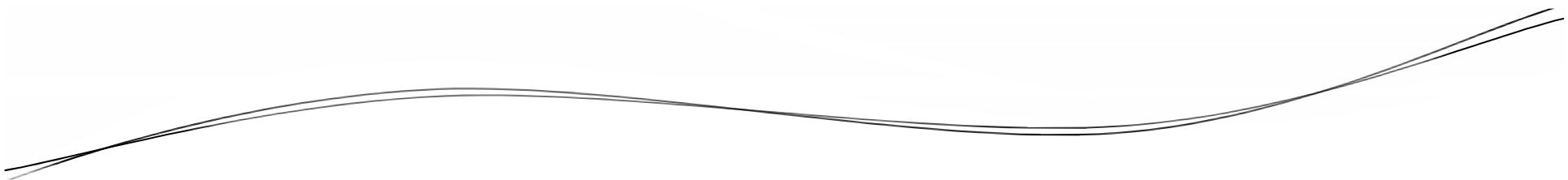
- It is also possible to handle multiple exceptions using a single catch block.
- This feature has been introduced by Java version 1.7.



Handling Multiple Exceptions

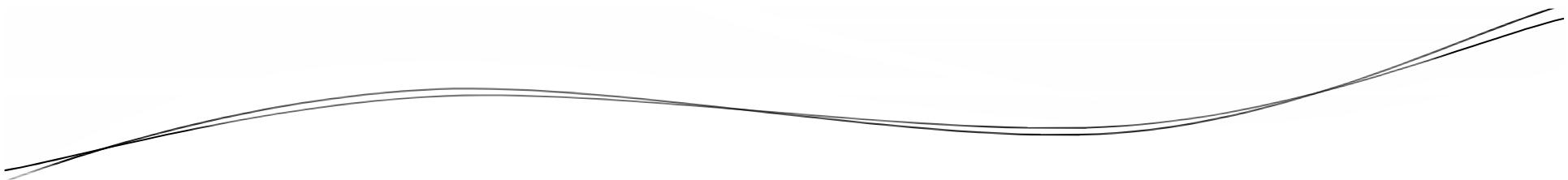
- E.g.

```
try {  
    //Statements  
}  
  
catch(<ex 1> | <ex 2> <ref-name>) {  
    //Statements  
}
```



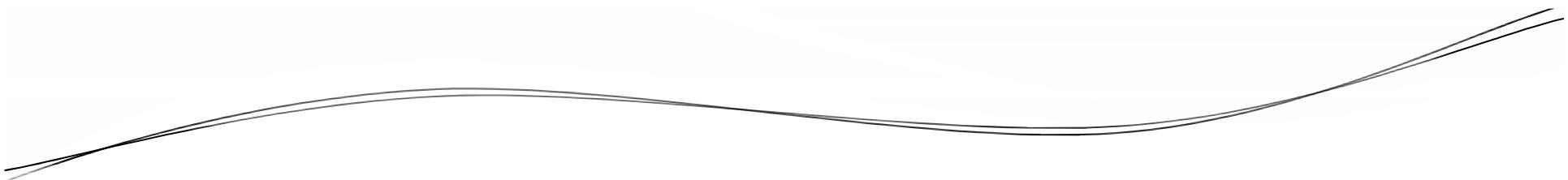
Handling Multiple Exceptions

- Since a single catch block is handling multiple exceptions, it is necessary to identify the type of the exceptions, so that different types of actions can be taken based upon the exception type.
- This is done by using instanceof operator.



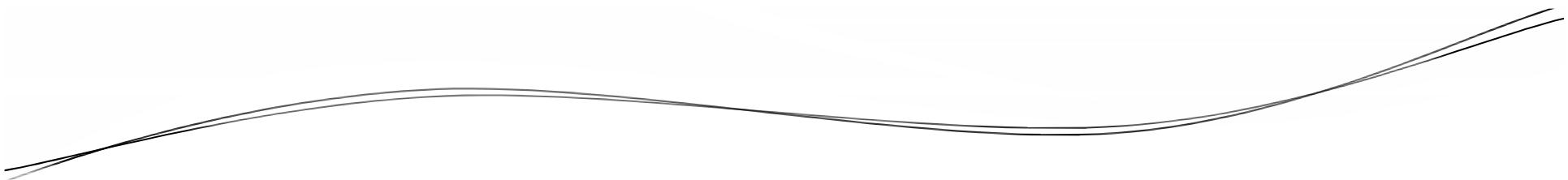
try / catch Limitation

By Rahul Barve



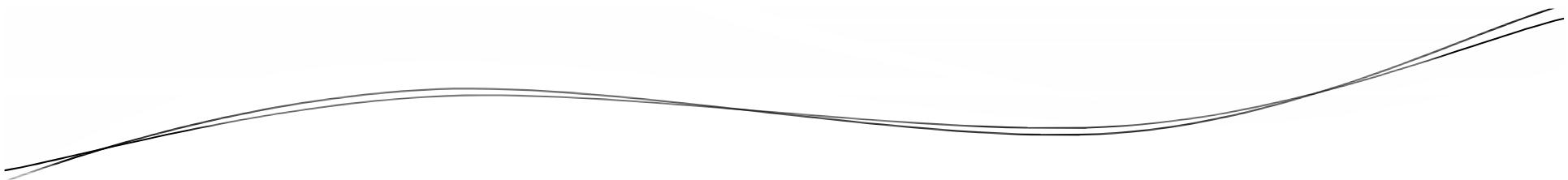
try / catch Limitation

- Although try and catch blocks are useful to handle the exceptions, they have a common limitation.
- None of these give guarantee about the execution of the statements.



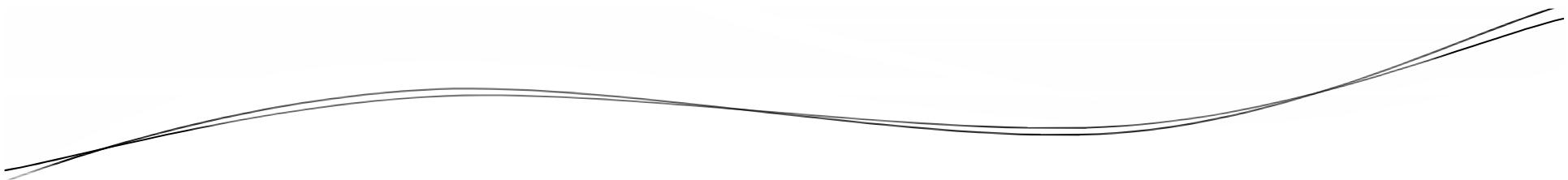
try / catch Limitation

- Sometimes, it becomes mandatory to execute the statements irrespective of whether the exception is fired or not.
- This is accomplished by using a finally block.



finally

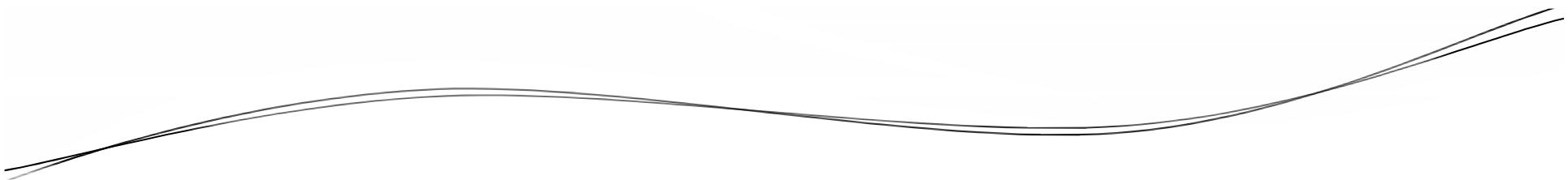
- Statements enclosed within a `finally` block always execute.
- This is generally useful to perform clean-up operations.



finally

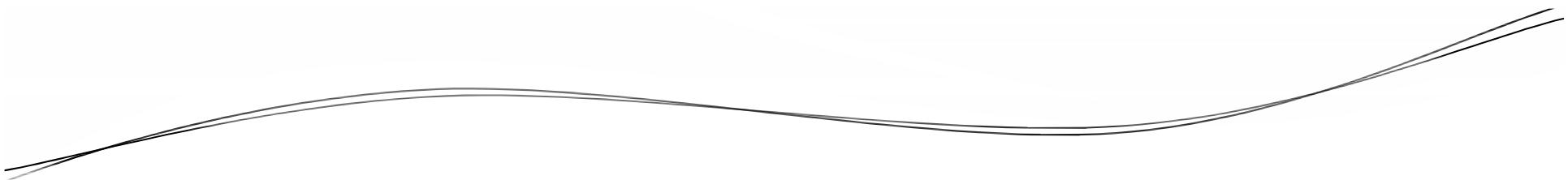
- E.g.

```
finally {  
    //Statements  
}
```



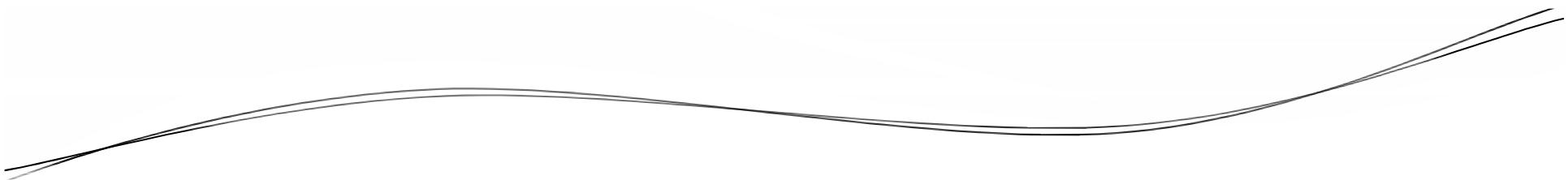
finally

- The `finally` block especially creates an impact for the methods of which the return type is other than `void`.



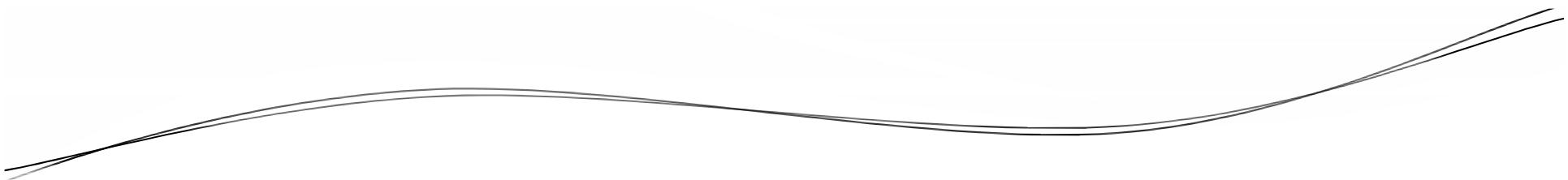
try-catch-finally Rules

By Rahul Barve



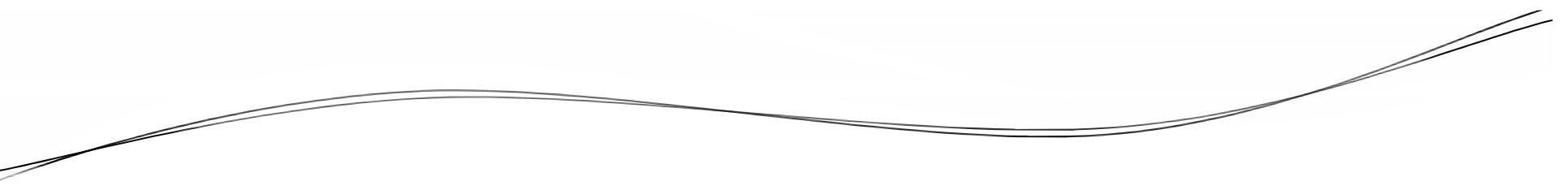
try-catch-finally Rules

- Every try block must be used in conjunction with either catch, finally or both.
- The blocks must appear one after the other without any statements in between.



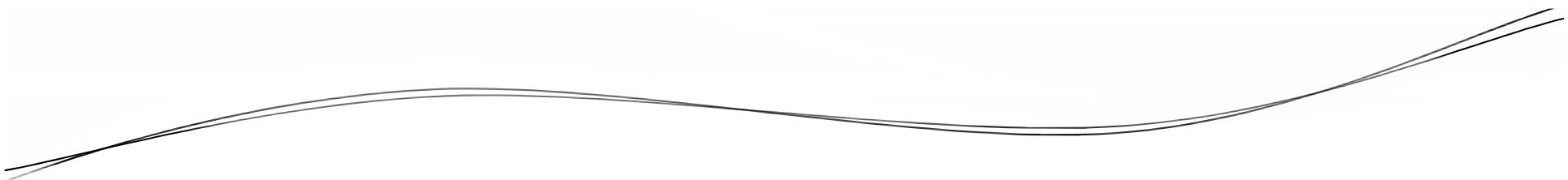
try-catch-finally Rules

- catch block cannot appear without try block.
- finally block cannot appear without try block.



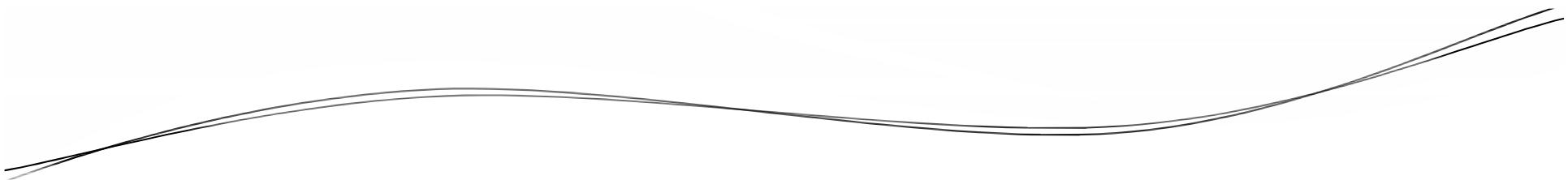
Using throws

By Rahul Barve



Using **throws**

- If several methods of a class are probable to fire an exception, it becomes difficult to manage writing try-catch constructs in each method.
- This can be simplified by using throws.

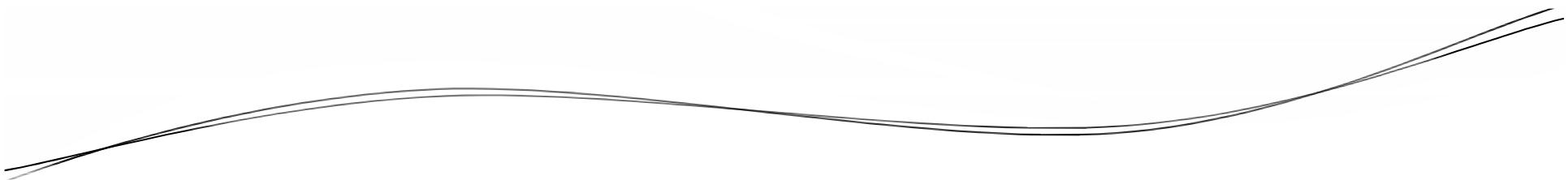


Using **throws**

- Used by method and constructor definitions which may fire exceptions but not willing to handle.
- Instructs the compiler to enforce the calling program to handle the exception (Checked Exceptions only).

Using **throws**

```
public void readFile(String fileName) throws  
FileNotFoundException {  
    //Statements  
}  
public void openFile(String fileName) {  
    readFile(fileName); //ERROR  
}
```



Using **throws**

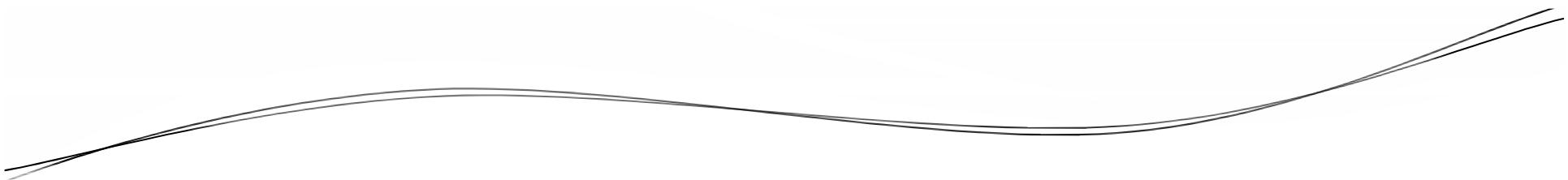
- If a called program method uses `throws` clause for a checked exception, then the calling program method, when using `throws` clause, must use a type which is equal or a super type.

Using **throws**

```
public class MyClass{  
    public void testOne()  
    throws Exception { ... }  
}
```

Using throws

```
public class YourClass {  
    //ERROR  
    public void testTwo() throws IOException{  
        new MyClass().testOne();  
    }  
    //OK  
    public void testThree() throws Exception{  
        new MyClass().testOne();  
    }  
    //OK  
    public void testFour() throws Throwable{  
        new MyClass().testOne();  
    }  
}
```

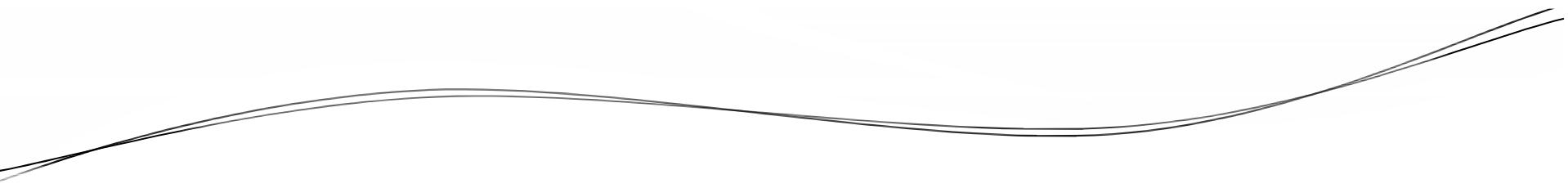


Using throws

- In method overriding, using throws clause, an overridden method can widen the scope but cannot narrow it.

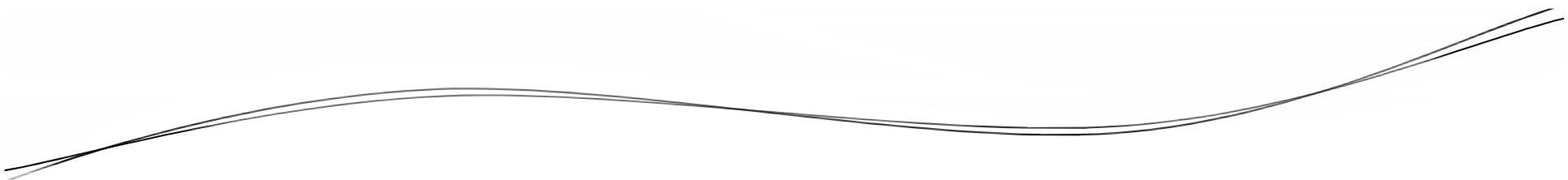
Using **throws**

```
public class Base {  
    public void test1() throws Exception {...}  
    public void test2() throws Exception {...}  
    public void test3() throws Exception {...}  
}  
  
public class Derived extends Base {  
    //OK  
    public void test1() throws IOException{...}  
    //OK  
    public void test2() {...}  
    //ERROR  
    public void test3() throws Throwable{...}  
}
```



Using `throw`

By Rahul Barve



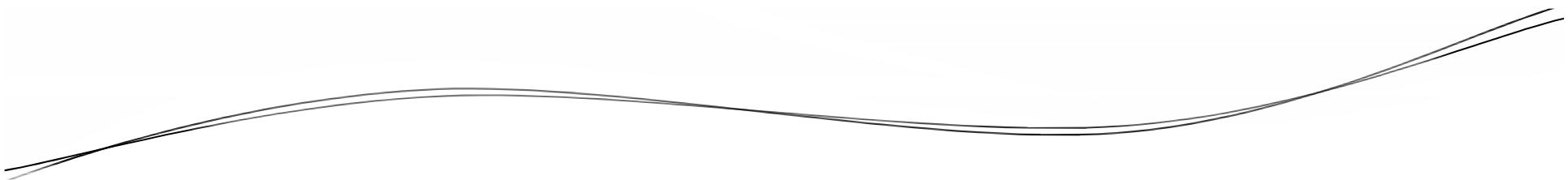
Using **throw**

- In most cases, JRE is responsible for firing an exception; but sometimes it might be necessary to fire an exception forcefully.
- This can be accomplished by using `throw` clause.

Using **throw**

- Syntax: `throw <Throwable>`
- E.g.

```
if(<condition>) {  
    Exception ex = new Exception();  
    throw ex;  
}
```



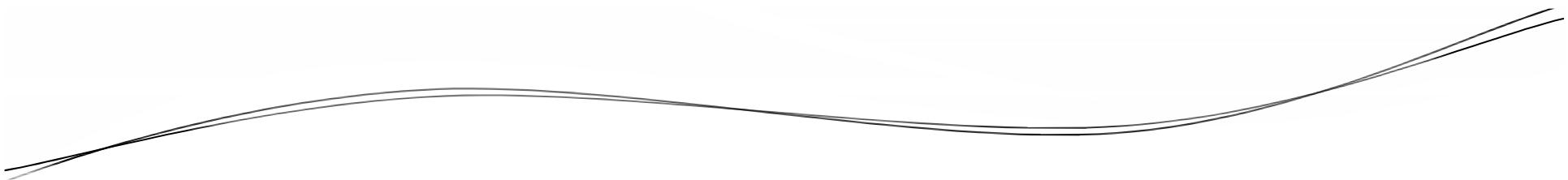
Using **throw**

- Sometimes, it becomes necessary to create a domain specific exception and throw it explicitly.
- Such exceptions are known as User Defined exceptions.

Using **throw**

- User defined exceptions are generally customized by creating a class that inherits either Exception or RuntimeException.
- E,g,

```
public class LowBalanceException  
extends Exception { ... }
```

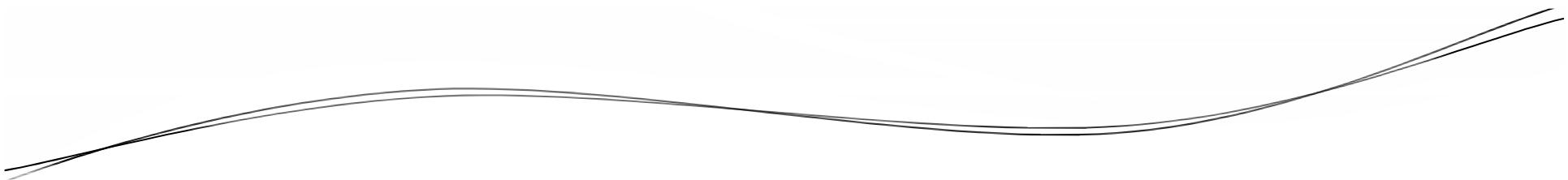


Using **throw**

- Once, a user defined exception class is created it can be used to raise an exception forcefully depending upon the condition.

Using **throw**

```
public void withdraw(float amount)
throws LowBalanceException {
    if(balance < amount) {
        String msg = "Low Balance!!";
        LowBalanceException lx =
            new LowBalanceException(msg);
        throw lx;
    }
}
```



Lets Summarize

- Introduction to Exception Handling
- Need for Exception Handling
- Exception Hierarchy
- Types of Exceptions
- Exception Handling Keywords
- User Defined Exceptions

```
1 package exceptions;  
2  
3 public class WithoutExceptionHandler {  
4  
5     public static void main(String[] args) {  
6         int n1 = Integer.parseInt(args[0]);  
7         int n2 = Integer.parseInt(args[1]);  
8         int result = n1 + n2;  
9         System.out.println("Result: " + result);  
10    }  
11}  
12  
13}  
14
```

```
WithoutExceptionHandlerExample.java X WithExceptionHandlerExample.java X MultipleExceptionHandlerExample.java X DefaultExceptionHandlerExample.java X ThrowsExample.java X ThrowExample.java X NameNotFoundException.java X
1 package exceptions;
2
3 public class WithExceptionHandlerExample {
4
5     public static void main(String[] args) {
6         int n1, n2, result = 0;
7         try {
8
9             n1 = Integer.parseInt(args[0]);
10            n2 = Integer.parseInt(args[1]);
11            result = n1 + n2;
12            System.out.println("Result: " + result);
13        }
14        catch(ArrayIndexOutOfBoundsException ex) {
15            //Here reference 'ex' refers to the exception class object that is created by JRE
16            System.out.println("Enter at least 2 numbers.");
17        }
18    }
19
20}
21
22}
23}
```

WithoutExceptionHandlerExample.java X WithExceptionHandlerExample.java X MultipleExceptionHandlerExample.java X DefaultExceptionHandlerExample.java X ThrowsExample.java X ThrowExample.java X NameNotFoundException.java X

```
1 package exceptions;
2
3 public class MultipleExceptionHandlerExample {
4
5     public static void main(String[] args) {
6         int n1, n2, result = 0;
7         try {
8
9             n1 = Integer.parseInt(args[0]);
10            n2 = Integer.parseInt(args[1]);
11            result = n1 / n2;
12            System.out.println("Result: " + result);
13        }
14        catch(ArrayIndexOutOfBoundsException ex) {
15            //Here reference 'ex' refers to the exception class object that is created by JRE
16            System.out.println("Enter at least 2 numbers.");
17        }
18        catch(ArithmeticException ex) {
19            System.out.println("Enter 2nd number as non-zero");
20        }
21
22    }
23
24
25 }
26 }
```

```
WithoutExceptionHandlerExample.java | WithExceptionHandlerExample.java | MultipleExceptionHandlerExample.java | DefaultExceptionHandlerExample.java | ThrowsExample.java | ThrowExample.java | NameNotFoundException.java | 
1 package exceptions;
2
3 public class DefaultExceptionHandlerExample {
4
5     public static void main(String[] args) {
6         int n1, n2, result = 0;
7         try {
8
9             n1 = Integer.parseInt(args[0]);
10            n2 = Integer.parseInt(args[1]);
11            result = n1 / n2;
12            System.out.println("Result: " + result);
13        }
14
15        /*catch(ArithmetricException ex) {
16            System.out.println("Enter 2nd number as non-zero");
17        }
18        catch(ArrayIndexOutOfBoundsException ex) {
19            //Here reference 'ex' refers to the exception class object that is created by JRE
20            System.out.println("Enter at least 2 numbers.");
21        }*/
22        //The above exception handlers can be defined by using a single 'catch' block
23        //This is a feature since Java 1.7
24        catch(ArithmetricException | ArrayIndexOutOfBoundsException ex) {
25            String message = "Enter at least 2 numbers.";
26            if(ex instanceof ArithmetricException)
27                message = "Enter 2nd number as non-zero";
28            System.out.println(message);
29        }
30    }
31}
```

WithoutExceptionHandlerExample.java WithExceptionHandlerExample.java MultipleExceptionHandler.java DefaultExceptionHandler.java ThrowsExample.java ThrowExample.java NameNotFoundException.java

```
25     String message = "Enter at least 2 numbers.";
26     if(ex instanceof ArithmeticException)
27         message = "Enter 2nd number as non-zero";
28     System.out.println(message);
29 }
30 catch(RuntimeException rx) {
31     System.out.println("Catching some runtime exception");
32 }
33 catch(Exception ex) {
34     System.out.println("This is a generic exception");
35 }
36 catch(Throwable t) {
37     System.out.println("Handling throwable");
38 }
39 // 'catch' block of sub type must appear before the 'catch' block of super type.
40
41 // Adding a 'finally' block
42 finally {
43     System.out.println("Thank you!!!");
44 }
45
46 }
47
48 }
49 }
```

```
1 package exceptions;
2
3 public class ThrowsExample {
4     private static void conductTest() throws Exception {
5         //This method instructs the compiler that it may throw an exception: Exception but not
6         //willing to handle it. Wants its caller to handle it
7     }
8
9     private static void doTest() {
10        //Invokes conductTest();
11        //This method is supposed to handle the exception raised by conductTest and it is handling that
12        try {
13            conductTest();
14        }
15        catch(Exception ex) {
16            //Some code
17        }
18    }
19    private static void performTest() throws Exception {
20        //This method is supposed to handle the exception raised by conductTest but it also does not
21        //want to handle it. Wants its caller to handle it
22        conductTest();
23    }
24
25    private static void doSimpleTest() throws RuntimeException {
26    }
27
28    public static void main(String[] args) {
```

```
WithoutExceptionHandlerExample.java WithExceptionHandlerExample.java MultipleExceptionHandlerExample.java DefaultExceptionHandlerExample.java ThrowsExample.java ThrowExample.java NameNotFoundException.java
25     private static void doSimpleTest() throws RuntimeException {
26
27 }
28     public static void main(String[] args) {
29         // TODO Auto-generated method stub
30         //Invoking doTest()
31         doTest();
32         //Invoking performTest()
33         //Here since performTest() further delegates the responsibility to its caller which is main(),
34         //the main() is supposed to handle it else delegate it.
35         //Since main() is the entry point, as per the standard practice, it must not add 'throws'
36         try {
37             performTest();
38         }
39         catch(Exception ex) {
40             //Some code
41         }
42
43         //Invoking doSimpleTest()
44         doSimpleTest();
45         //Since this method is capable of firing RuntimeException which is an unchecked exception
46         //'throws' does not create any impact
47
48     }
49
50 }
51
52 }
```

```
WithoutExceptionHandlerExample.java X WithExceptionHandlerExample.java X MultipleExceptionHandlerExample.java X DefaultExceptionHandlerExample.java X ThrowsExample.java X ThrowExample.java X NameNotFoundException.java X
1 package exceptions;
2
3 public class ThrowExample {
4     private static String greetUser(String name, String message) {
5         //This method greets the user with some message if the name contains at least 5 characters
6         //Otherwise fires an exception RuntimeException
7         int length = name.length();
8         if(length < 5) {
9             RuntimeException rx = new RuntimeException("The name length is invalid...");
10            throw rx;
11        }
12        String greeting = message + " " + name;
13        return greeting;
14    }
15    public static void main(String[] args) {
16        try {
17            String msg = greetUser("Jack", "Welcome");
18            System.out.println(msg);
19        }
20        catch(RuntimeException rxe) {
21            String errorMessage = rxe.getMessage();
22            System.out.println(errorMessage);
23        }
24    }
25 }
26
27
28 }
```

WithExceptionHandlerExample.java X MultipleExceptionHandlerExample.java X DefaultExceptionHandlerExample.java X ThrowsExample.java X ThrowExample.java X NameNotFoundException.java X NameCollection.java X UserDefinedE

```
1 package exceptions;
2
3 public class NameNotFoundException extends Exception{
4     private String invalidName;
5     public NameNotFoundException(String errorMessage, String invalidName) {
6         super(errorMessage);
7         this.invalidName = invalidName;
8     }
9
10    @Override
11    public String getMessage() {
12        String errMessage = super.getMessage();
13        String finalMessage = errMessage + ": " + invalidName;
14        return finalMessage;
15    }
16
17
18 }
19 }
```

```
1 package exceptions;
2
3 public class NameCollection {
4     private static String[] names = {"Govinda", "Ranbir", "Ranveer", "Alia", "Deepika"};
5
6     public static int getPosition(String name) throws NameNotFoundException {
7         int position = -1;
8         int length = names.length;
9         for(int index = 0; index < length; index++) {
10             String currentName = names[index];
11             if(currentName.equals(name)) {
12                 position = index;
13             }
14         }
15         if(position == -1) {
16             NameNotFoundException nx =
17                 new NameNotFoundException("The supplied name is invalid", name);
18             throw nx;
19         }
20         return position;
21     }
22 }
23 }
```

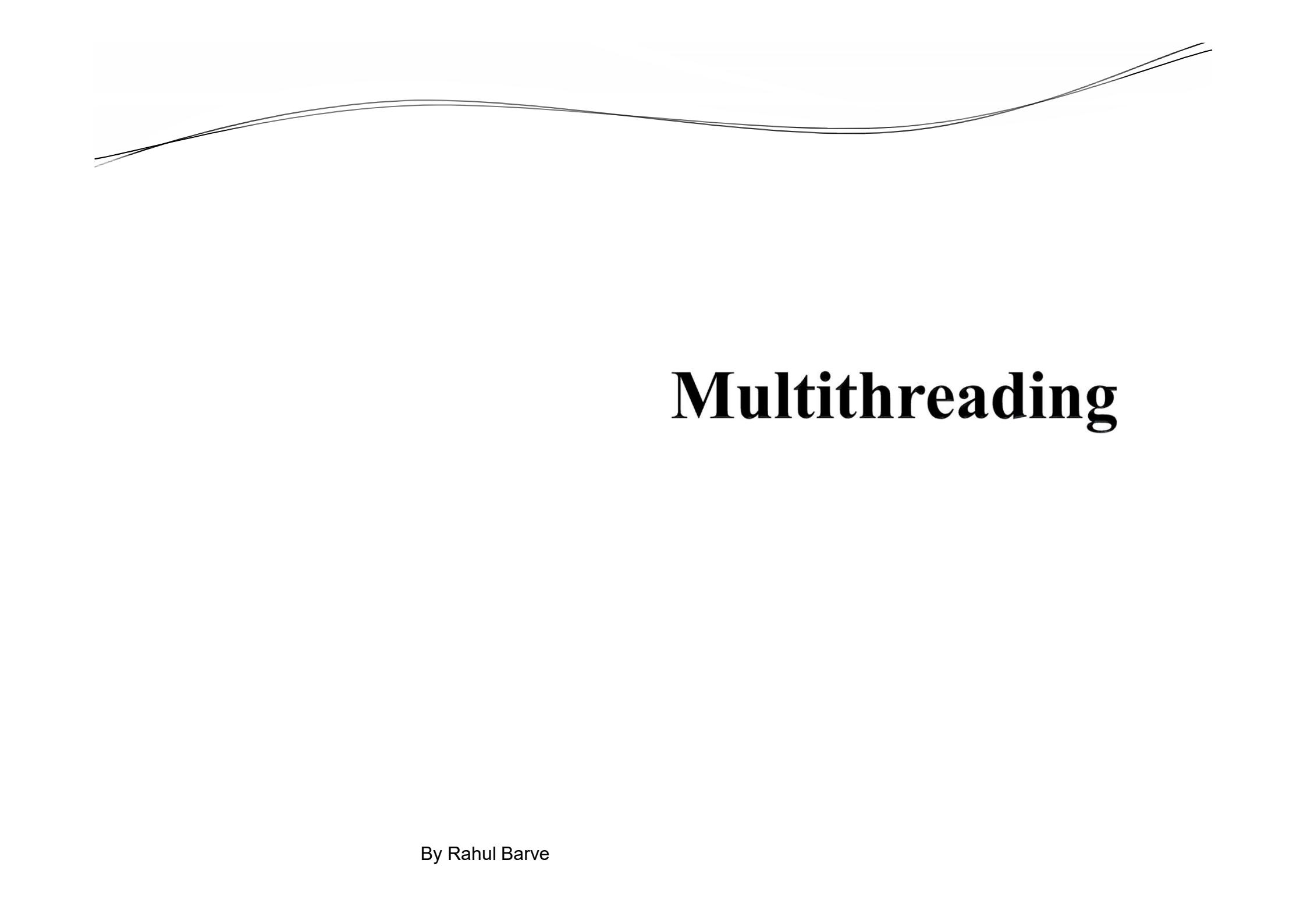
```
1 package exceptions;
2
3 public class UserDefinedExceptionExample {
4
5     public static void main(String[] args) {
6         try {
7             int position = NameCollection.getPosition("Govinda");
8             System.out.println("Position is " + position);
9         } catch (NameNotFoundException e) {
10             // TODO Auto-generated catch block
11             //e.printStackTrace();
12             System.out.println(e.getMessage());
13         }
14     }
15
16 }
17
18
19 }
```

DefaultExceptionHandlerExample.java ✘ ThrowsExample.java ✘ ThrowExample.java ✘ NameNotFoundException.java ✘ NameCollection.java ✘ UserDefinedExceptionExample.java ✘ Assignment 7.txt ✘

```
1 Time Limit: 45 Minutes
2
3 Create a class Person with following attributes:
4     name (String)
5     emailAddress (String)
6     age (int)
7
8 Create user defined exception classes as per the following:
9     InvalidEmailAddressException
10    InvalidAgeException
11
12 Write a main program to create 2 objects of a Person class and display the information.
13 The program must throw 1st exception if the email address does not contain '@' and '.'
14 characters and 2nd exception if the age is below 18.
15
16 Display appropriate error messages.
17 (Note: Both the exceptions must be checked exceptions)
18 (Use String class contains() method)
19
20
21
```

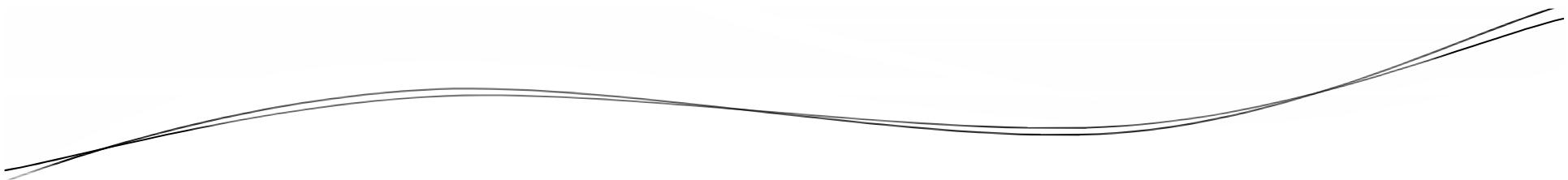
```
16      is the name of the operation:  
17      ADD, SUBTRACT, MULTIPLY, DIVIDE  
18  
19      The program must perform the following:  
20      A) Handle ArrayIndex... exception for insufficient  
         values  
21      B) Handle Arithmetic...exception in case of  
         division operation  
22      C) Throw an exception of type Exception with some  
         message if the operation name is invalid  
23         and catch that exception appropriately.  
24      -----  
25 2.  
26 2.  
27 2.  
28 2.  
29 Modify the Shirts assignment as per the following:  
30 Create a user defined exception: ShirtTypeNotFoundException  
31  
32 Modify the method printAllShirts(String type) in such a way  
33 that it fires the exception ShirtTypeNotFoundException if the  
34 Shirt with the specified type is not found.  
35  
36 This method must insist the caller to handle the exception.  
37  
38  
39  
40
```

```
DefaultExceptionHandlerExample.java ✘ ThrowsExample.java ✘ ThrowExample.java ✘ NameNotFoundException.java ✘ NameCollection.java ✘ UserDefinedExceptionExample.java ✘ Assignment 7.txt ✘ Exception Assignments.txt ✘
1 Extra Assignments on Exception Handling:
2
3 1.
4 Create a class MathOperation with following attributes:
5     operation (String) e.g. "Addition", "Subtraction" etc
6     firstNumber (int)
7     secondNumber (int)
8
9     provide an instance method perform() to perform
10    one of the 4 mathematical operations based upon
11    the 'operation' value.
12
13    Write a main program that accepts 3 command line
14    arguments:
15    1st 2 arguments are 2 numbers and the 3rd argument
16    is the name of the operation:
17    ADD, SUBTRACT, MULTIPLY, DIVIDE
18
19    The program must perform the following:
20    A) Handle ArrayIndex... exception for insufficient
21        values
22    B) Handle Arithmetic...exception in case of
23        division operation
24    C) Throw an exception of type Exception with some
25        message if the operation name is invalid
26        and catch that exception appropriately.
27 -----
28 2.
29 Modify the Shirts assignment as per the following:
30 Create a user defined exception: ShirtTypeNotFoundException
31
32 Modify the method printAllShirts(String type) in such a way
```



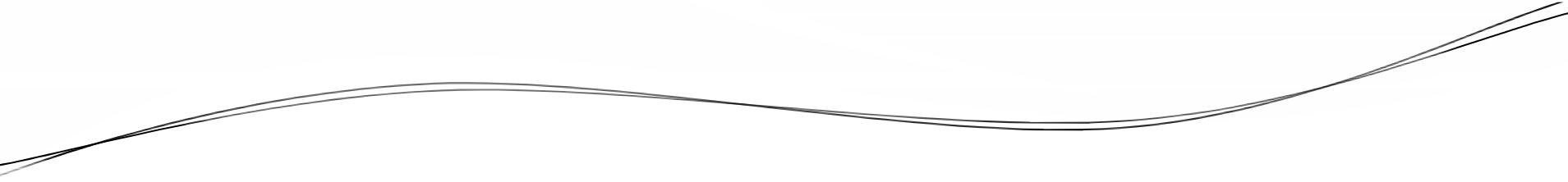
Multithreading

By Rahul Barve



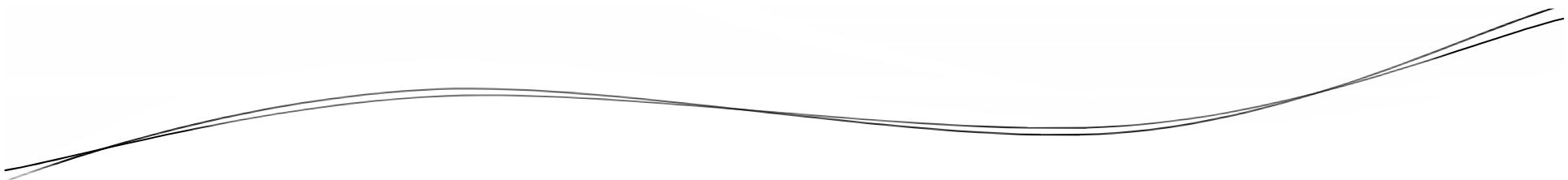
Objectives

- Understand Multitasking
- Introduction to Multithreading
- Need for Multithreading
- Implementing Multithreading
- Thread Life Cycle
- Thread Priorities
- Understanding Thread Methods
- Synchronization
- Inter-thread Communication



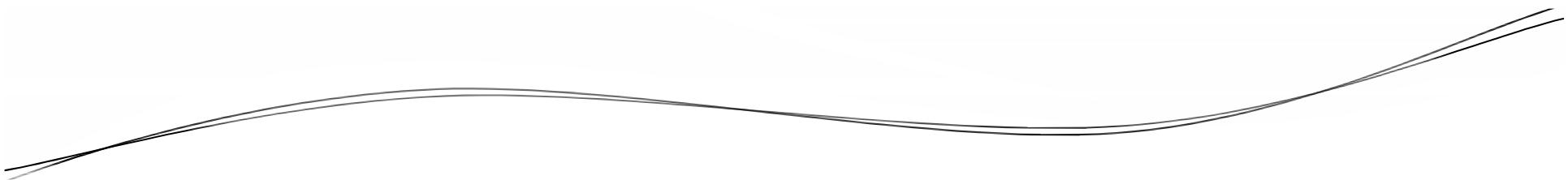
Multitasking

By Rahul Barve



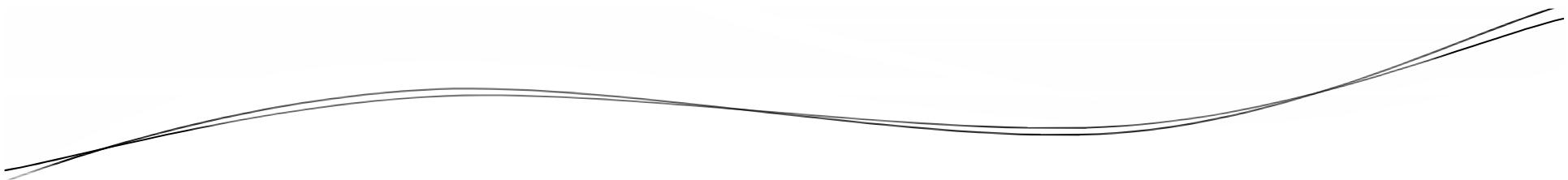
Multitasking

- Multitasking is a process that involves multiple tasks running simultaneously.
- It is a generic terminology which has 2 forms: Multiprocessing and Multithreading.



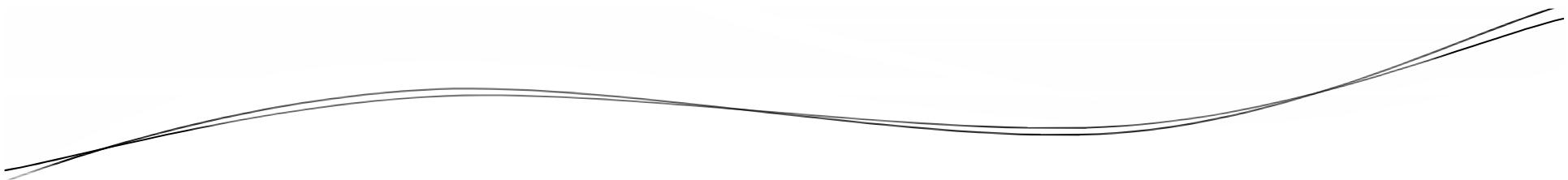
Multiprocessing

By Rahul Barve



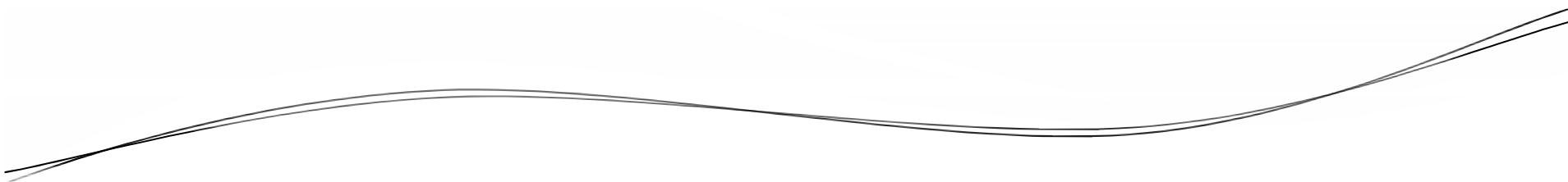
Multiprocessing

- Program is a set of instructions and a Process is a ‘Running Instance of a Program’.
- CPU is shared between 2 processes.



Multiprocessing

- In Multiprocessing, OS implements Context Switching mechanism.
- Program's entire context including variables, global variables, objects etc., is stored separately.



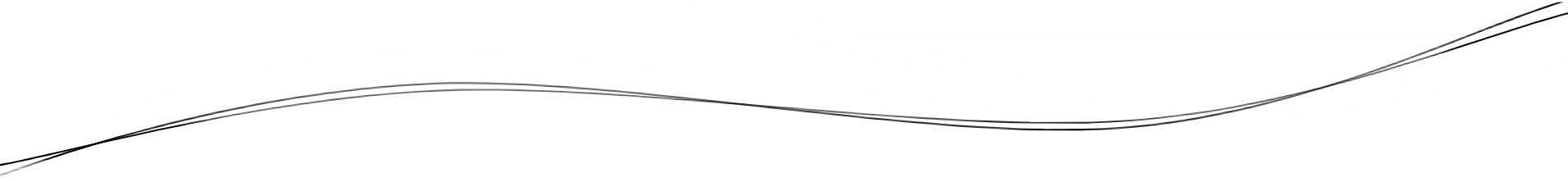
Multiprocessing

MS WORD

**Variables
Global Variables
Objects**

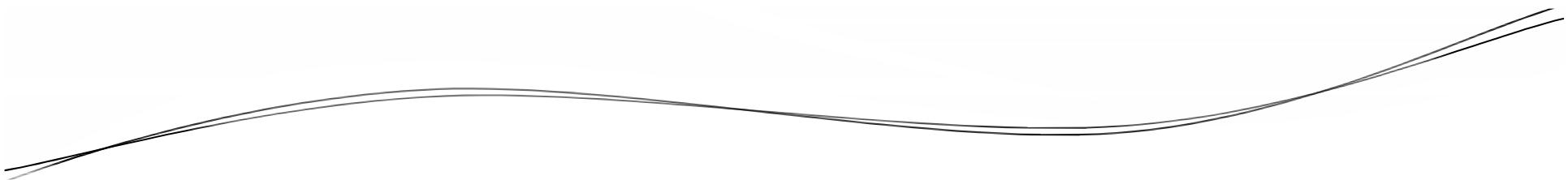
WINAMP

**Variables
Global Variables
Objects**



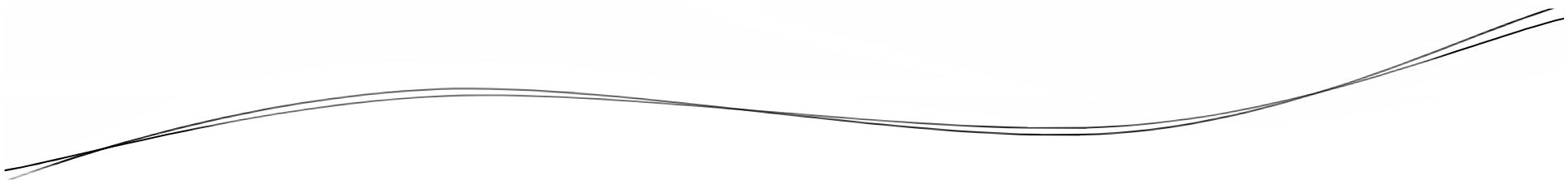
Multithreading

By Rahul Barve



Multithreading

- Different tasks within a main task execute simultaneously.
- Multithreading implements the idea of Multitasking by taking it one level lower.



Multithreading

- Each sub task within an application is called as a Thread.
- Since multiple threads run within a same application, may share the data.

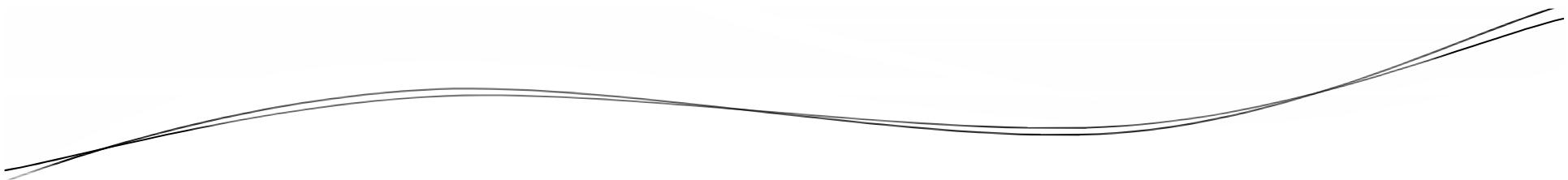
Multithreading

WINAMP

playSong()

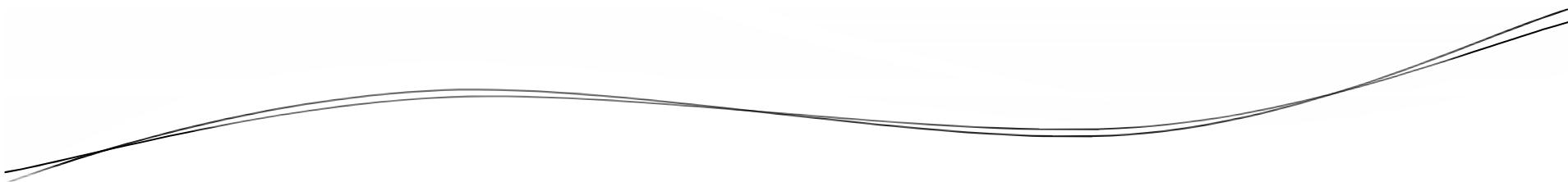
**Variables
Global Variables
Objects**

createPlayList()



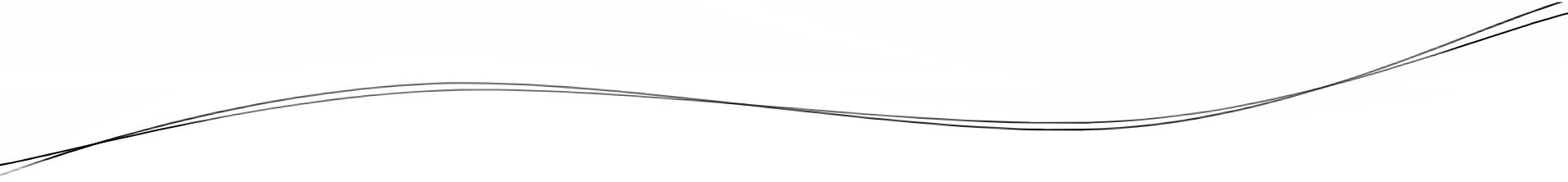
Multithreading

- A Thread is an entity within a Process.
- It defines the path of execution.



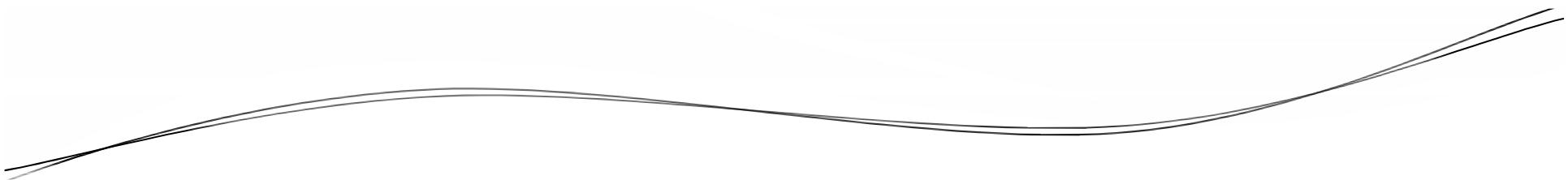
Process Vs. Thread

- Each process has a complete set of its own variables.
- It takes more overhead to launch a new process.
- Inter-process communication is a heavyweight activity.
- Threads live within a single process, thus may share the same data.
- It takes much less overhead to launch a new thread.
- Inter-thread communication is a lightweight activity.



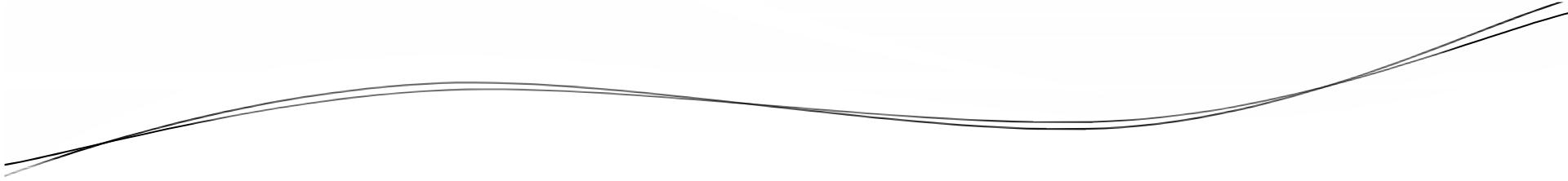
Implementing Multithreading

By Rahul Barve



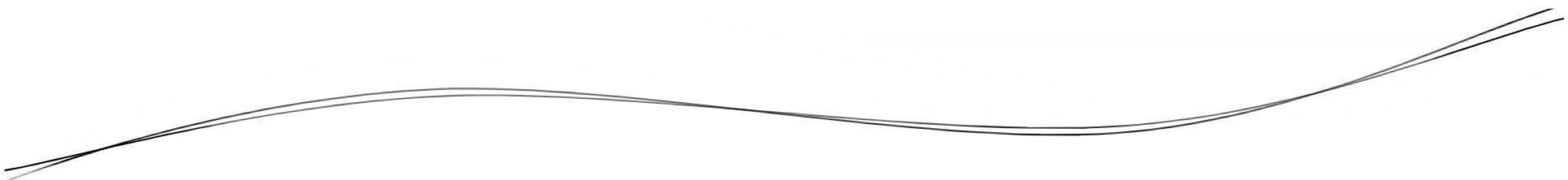
Implementing Multithreading

- In order to implement multithreading Java provides an API from `java.lang` package:
- A `Thread` class and a `Runnable` interface.



Implementing Multithreading

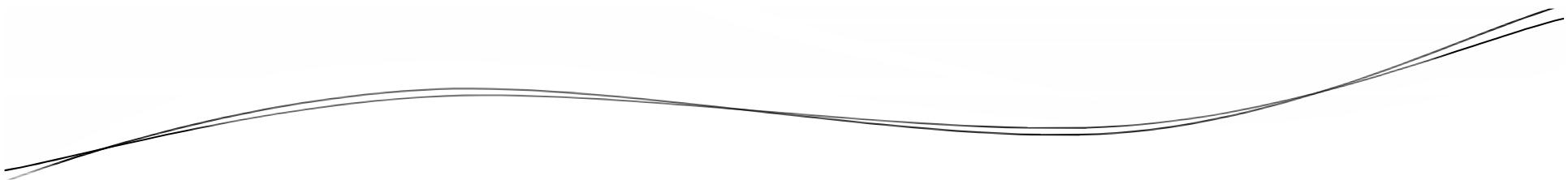
- To implement multithreading, it is necessary to create a class that is known as a Thread Implementation Class.
- It must either extend Thread or implement Runnable.



Implementing Multithreading

```
public class MyThread extends Thread {  
    public void run() {...}  
}
```

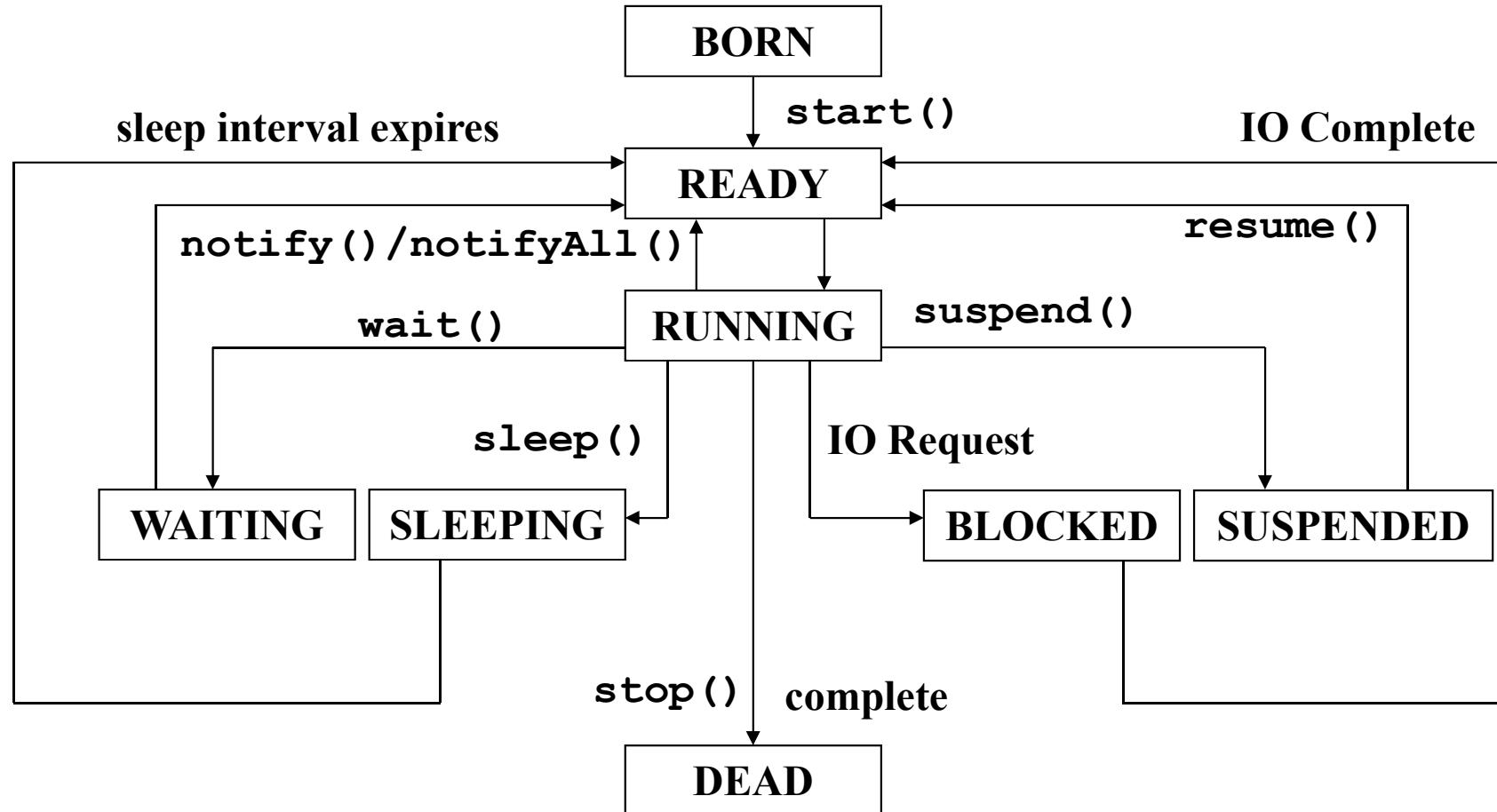
```
public class MyThread implements Runnable {  
    public void run() {...}  
}
```

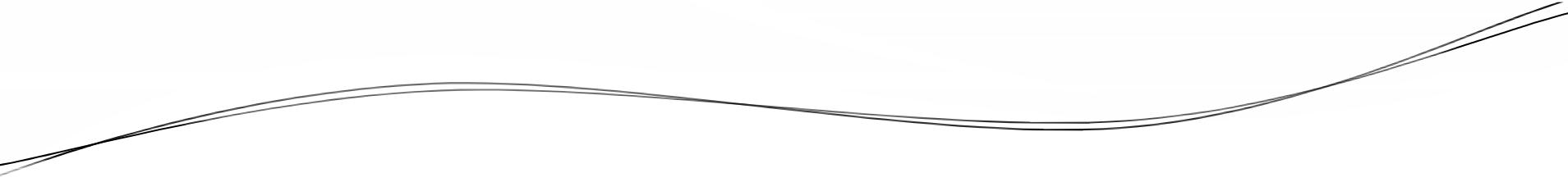


Implementing Multithreading

- All thread implementation classes define a `run()` method which provides a logic for the thread or acts as a gateway to the logic.

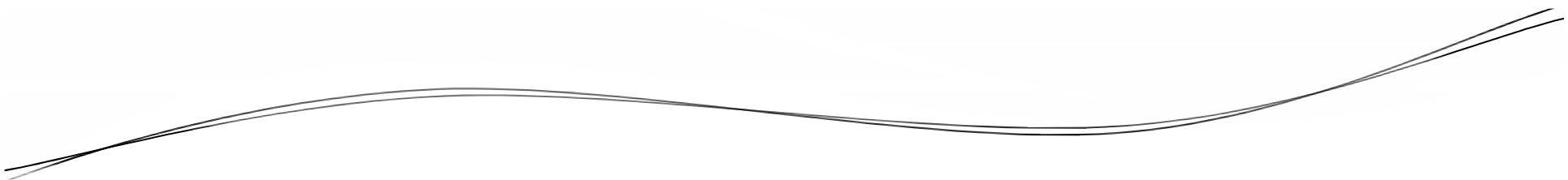
Life Cycle of Thread





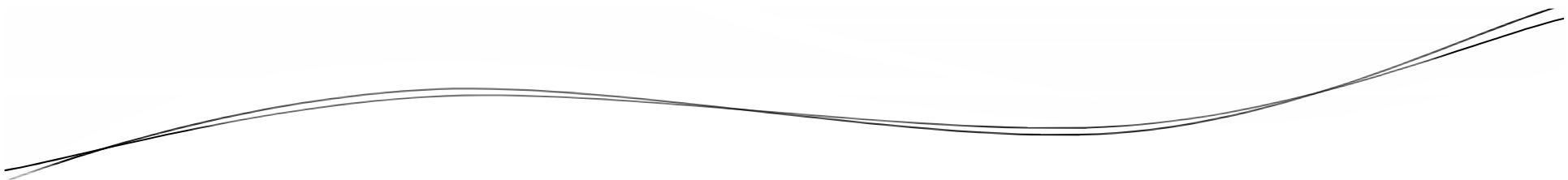
Thread Priorities

By Rahul Barve



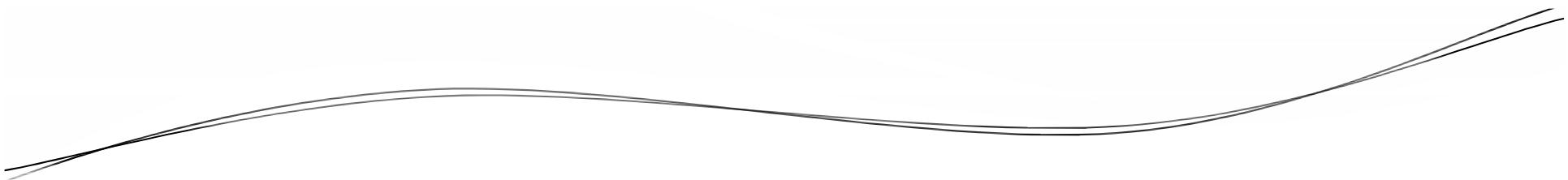
Thread Priorities

- Depending upon the task the thread is going to perform, it is possible to assign a priority to the thread.
- Priorities can be specified using `setPriority()` method within the range 1 to 10, being 1 as lowest and 10 as highest.



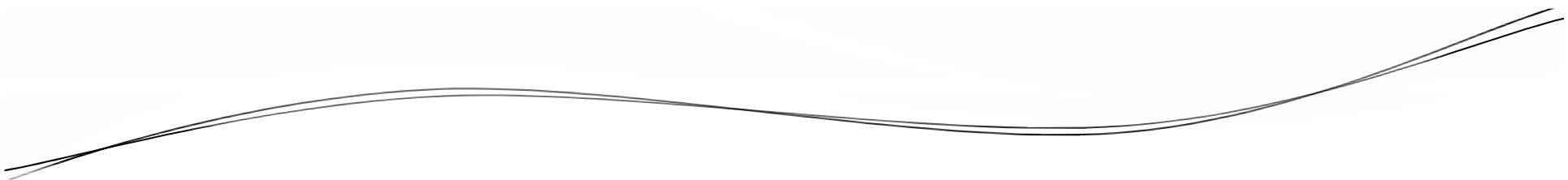
Thread Priorities

- The scheduler picks up the highest priority thread that is currently in the READY state.
- The default priority of a thread is 5.



Thread Methods

By Rahul Barve

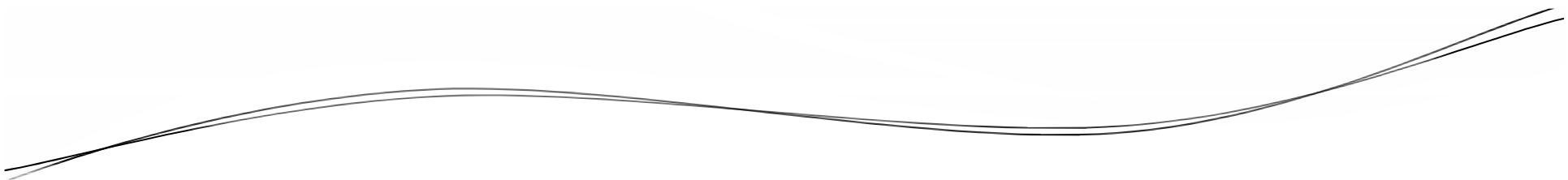


Thread Methods

- start()
- stop()
- yield()
- isAlive()
- sleep()
- suspend()
- resume()
- currentThread()
- join()

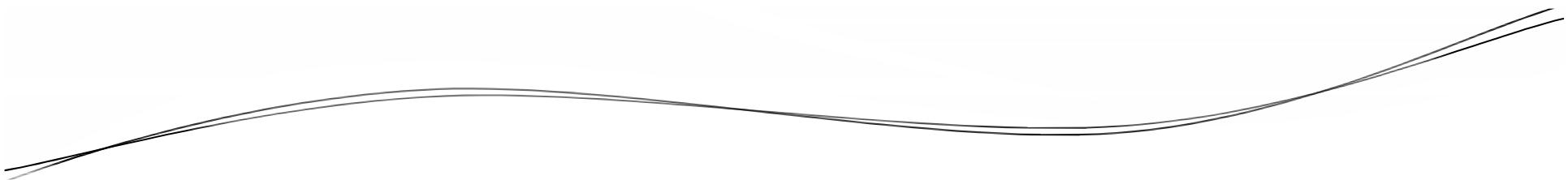
start()

- A method that makes a request to OS for the creation of the Thread.
- Responsible for transitioning of the thread from BORN state to READY state.



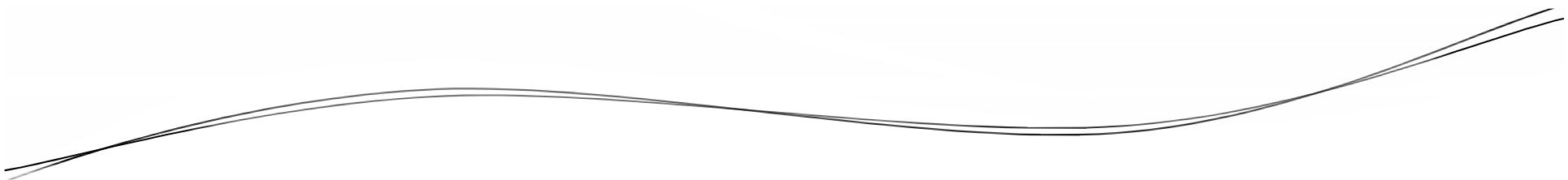
stop()

- Forcefully kills the thread.
- Sends the thread into DEAD state.
- Declared as deprecated.



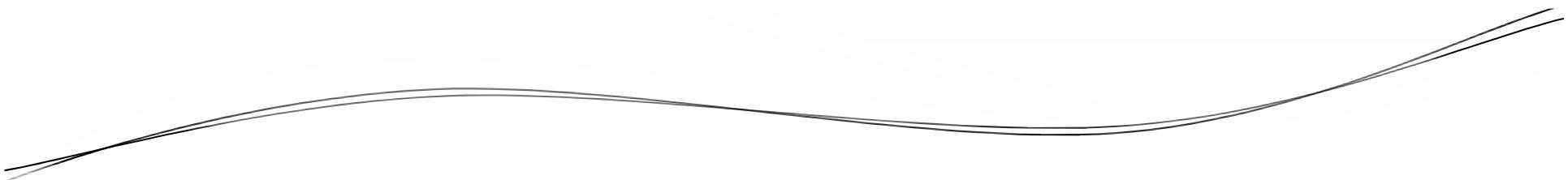
yield()

- A static method that causes currently executing thread to yield the control.
- If there are other runnable threads of which, priority is at least as high as this thread, they will be scheduled next.



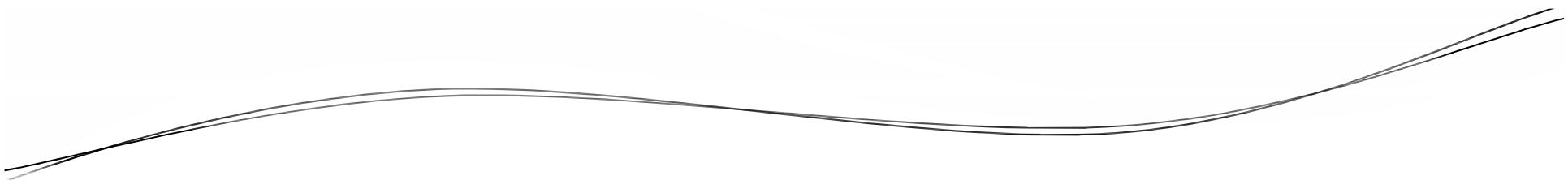
isAlive()

- Returns true if a thread has started and not yet terminated.



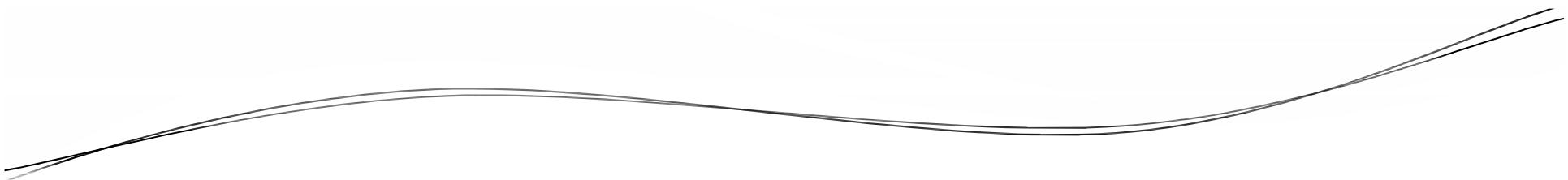
sleep()

- A static method that sends a thread into the SLEEPING state.
- A thread remains into the SLEEPING state until the sleep time interval is over.



suspend () and resume ()

- **suspend () :**
 - Sends a thread into the SUSPENDED state.
- **resume () :**
 - Brings the thread into the READY state.
- Both are declared as deprecated.

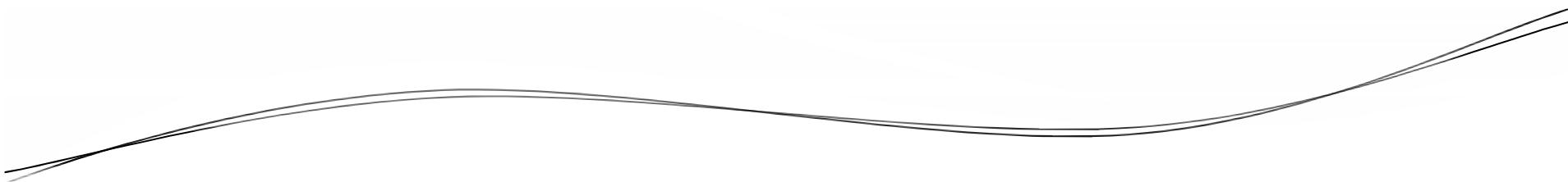


currentThread()

- A static method that returns a reference to the thread that is currently running.

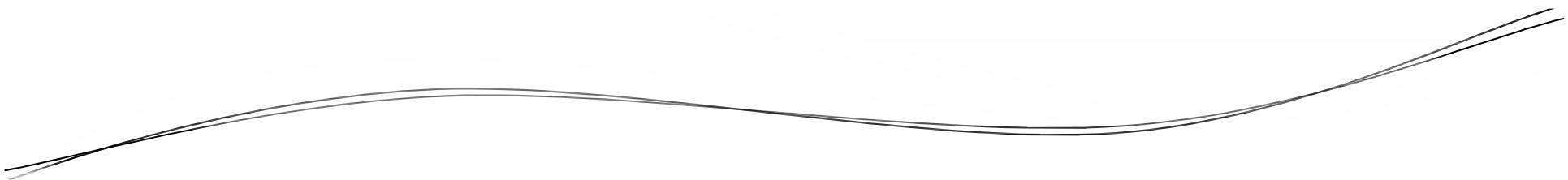
join ()

- Causes the parent thread to wait until the termination of the child thread on which it is invoked.



Synchronization

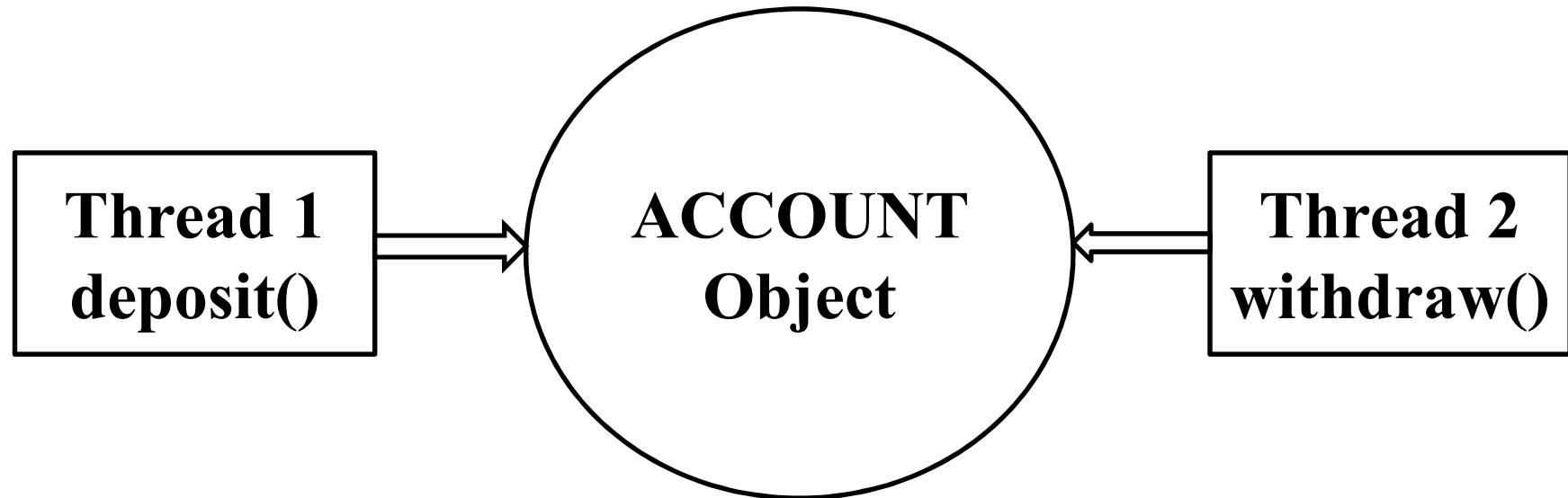
By Rahul Barve

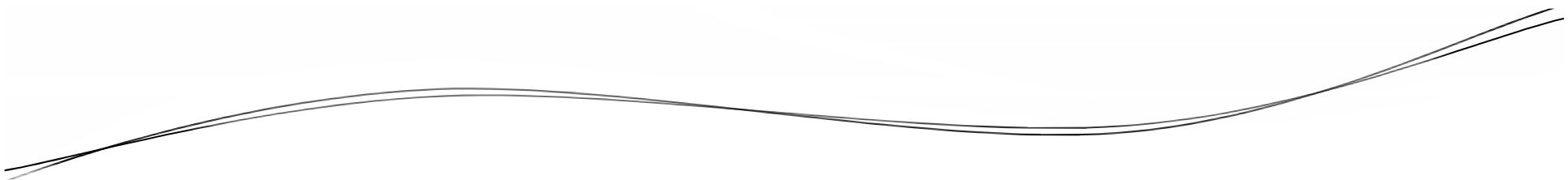


Synchronization

- Many times in an application, two or multiple threads need to access the same object.
- This needs to ensure that the object will be modified by only one thread at a time, otherwise they will fall into race condition.

Synchronization

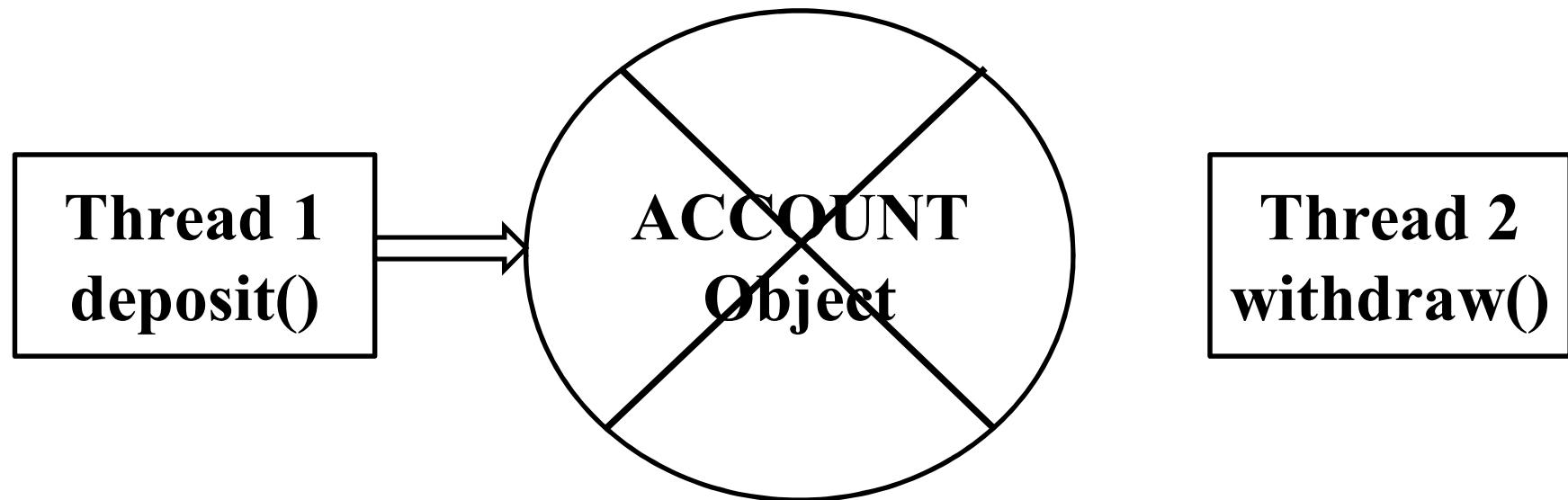


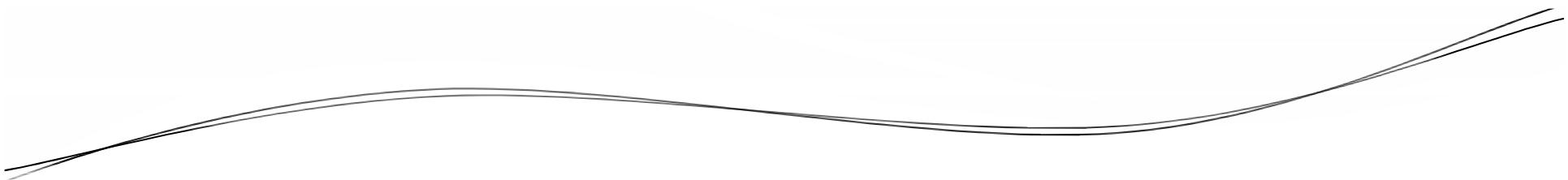


Synchronization

- Key to synchronization is the concept of monitor.
- A monitor is an object that is used as a mutually exclusive lock.
- Only one thread can own a monitor at a given time.

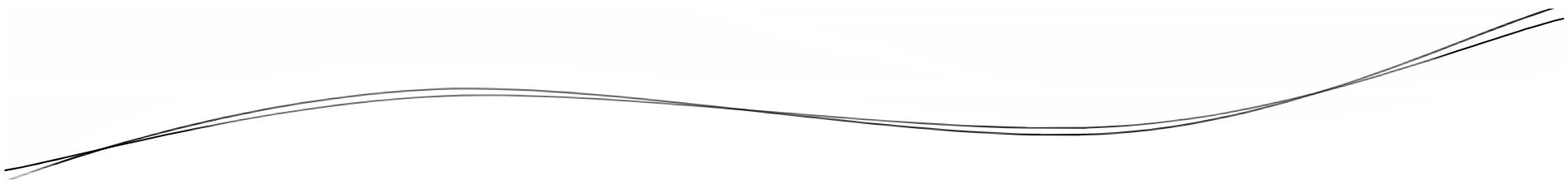
Synchronization





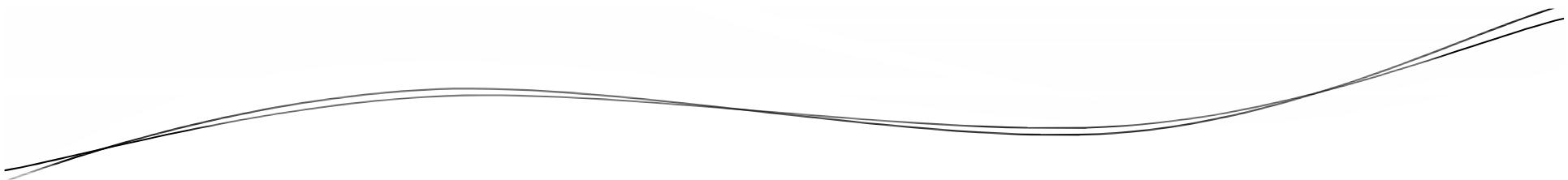
Synchronization

- When a thread acquires a lock, it is said to have entered the monitor.
- All other threads attempting to enter the locked monitor will get suspended, until the first thread exits the monitor.



Synchronization

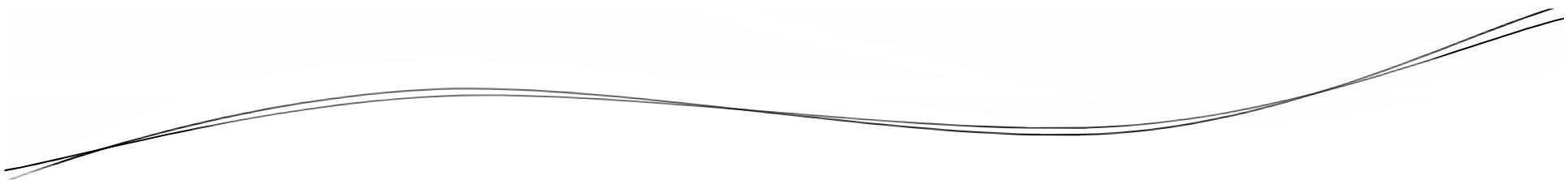
- Thread synchronization is achieved in 2 ways:
 - Using synchronized methods.
 - Using synchronized blocks.



Synchronization

- Using synchronized methods.

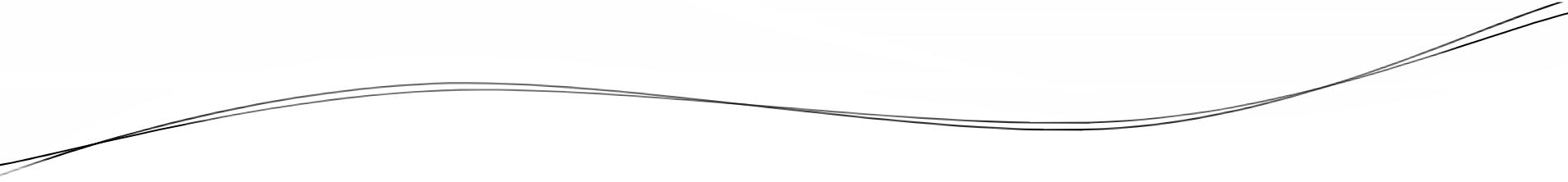
```
public synchronized void m1() {  
    //Some Code  
}
```



Synchronization

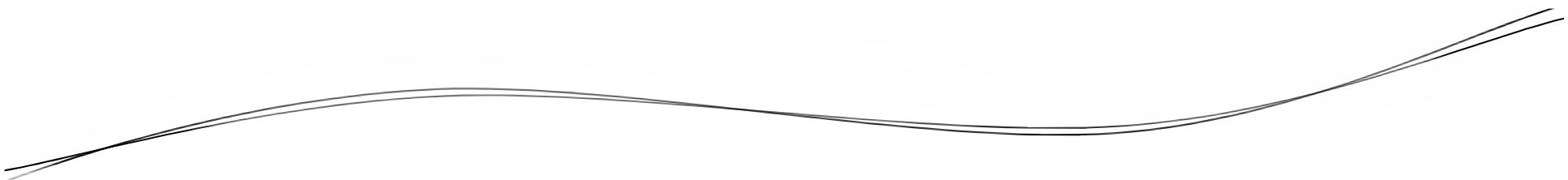
- Using synchronized blocks.

```
synchronized(obj) {  
    //Some Code  
}
```



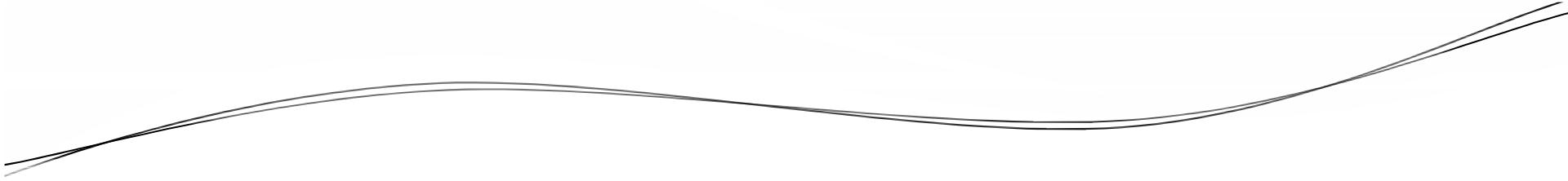
Inter-Thread Communication

By Rahul Barve



Inter-Thread Communication

- Sometimes, in an application there is a need of two or multiple threads interacting with each other. It leads to inter-thread communication.



Inter-Thread Communication

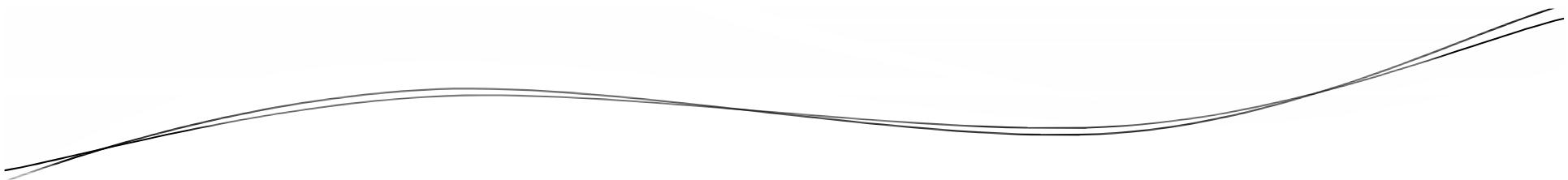
- To implement inter-thread communication, there are 3 methods used:
 - `wait()`
 - `notify()`
 - `notifyAll()`

wait()

- Must be invoked within a synchronized context.
- When invoked, releases the lock and sends the currently running thread into the WAITING state.
- Gives a chance to other threads looking for the same lock.

notify() & notifyAll()

- `notify()`
 - Wakes up a single thread that is in WAITING state.
- `notifyAll()`
 - Wakes up all threads.



Lets Summarize

- Multitasking
- Multithreading
- Need for Multithreading
- Implementing Multithreading
- Thread Life Cycle
- Thread Priorities
- Thread Methods
- Synchronization
- Inter-thread Communication

TestThread.java X BasicExample.java X GreetingThread.java X ThreadExample.java X RunnableExample.java X GreetingRunnable.java X CurrentThreadExample.java X GreetingRunnableUsingCurrentThread.java X GreetingRunnable...

```
1 package multithreading;
2 //This is the Thread implementation class
3 public class TestThread extends Thread{
4     public void run() {
5         //When thread execution begins, this method gets invoked
6         System.out.println("Test thread works !!");
7     }
8 }
9
10 }
```

TestThread.java X BasicExample.java X GreetingThread.java X ThreadExample.java X RunnableExample.java X GreetingRunnable.java X CurrentThreadExample.java X GreetingRunnableUsingCurrentThread.java X GreetingRunnab

```
1 package multithreading;
2
3 public class BasicExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Thread t = new TestThread();
8         t.start(); //Starts the thread
9
10    }
11
12 }
13
```

```
1 package multithreading;
2 //This is the Thread implementation class
3 public class GreetingThread extends Thread{
4     private String greetingMessage;
5     private int delayTime;//In Milliseconds
6
7     public GreetingThread(String greetingMessage, int delayTime) {
8         this.greetingMessage = greetingMessage;
9         this.delayTime = delayTime;
10    }
11
12    public void run() {
13        for(int a=1;a<=10;a++) {
14            System.out.println(greetingMessage);
15            //Introduce a time gap
16            try {
17                Thread.sleep(delayTime);//Sending the currently running thread into Sleeping State
18            } catch (InterruptedException e) {
19                // TODO Auto-generated catch block
20                e.printStackTrace();
21            }
22        }
23    }
24
25}
26
27
28
29
30
31
32
```

TestThread.java X BasicExample.java X GreetingThread.java X ThreadExample.java X RunnableExample.java X GreetingRunnable.java X CurrentThreadExample.java X GreetingRunnableUsingCurrentThread.java X GreetingRunnable X

```
1 package multithreading;
2
3 public class ThreadExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Thread t1 = new GreetingThread("Hello", 2000);
8         Thread t2 = new GreetingThread("Welcome", 1000);
9         t1.start();
10        t2.start();
11
12    }
13
14 }
15
```

TestThread.java | BasicExample.java | GreetingThread.java | ThreadExample.java | **RunnableExample.java** | GreetingRunnable.java | CurrentThreadExample.java | GreetingRunnableUsingCurrentThread.java | GreetingRunnab

```
1 package multithreading;
2
3 public class RunnableExample {
4
5     public static void main(String[] args) {
6         Runnable r1 = new GreetingRunnable("Hello", 2000);
7         Runnable r2 = new GreetingRunnable("Welcome", 1000);
8         //Setting Runnable as a target
9         Thread t1 = new Thread(r1);
10        Thread t2 = new Thread(r2);
11        t1.start();
12        t2.start();
13    }
14
15 }
16
17 }
```

```
package multithreading;
//This is the Thread implementation class
public class GreetingRunnable implements Runnable{
    private String greetingMessage;
    private int delayTime;//In Milliseconds

    public GreetingRunnable(String greetingMessage, int delayTime) {
        this.greetingMessage = greetingMessage;
        this.delayTime = delayTime;
    }

    public void run() {
        for(int a=1;a<=10;a++) {
            System.out.println(greetingMessage);
            //Introduce a time gap
            try {
                Thread.sleep(delayTime);//Sending the currently running thread into Sleeping State
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

TestThread.java X BasicExample.java X GreetingThread.java X ThreadExample.java X RunnableExample.java X GreetingRunnable.java X CurrentThreadExample.java X GreetingRunnableUsingCurrentThread.java X GreetingRunnable...

```
1 package multithreading;
2
3 public class CurrentThreadExample {
4
5     public static void main(String[] args) {
6         //Obtaining a handle to the currently running thread which is main thread
7         Thread mainThread = Thread.currentThread();
8         String name = mainThread.getName();
9         System.out.println("Main thread name is " + name);
10        //Stopping the main thread
11        mainThread.stop(); //Stops this thread hence the next line does not execute
12        System.out.println("This message will not be printed");
13    }
14
15 }
16
17 }
```

```
package multithreading;
//This is the Thread implementation class
public class GreetingRunnableUsingCurrentThread implements Runnable{

    public void run() {
        for(int a=1;a<=10;a++) {
            Thread currThread = Thread.currentThread();
            String name = currThread.getName();
            if(name.equals("first")) {
                System.out.println("Hello");
                try {
                    Thread.sleep(1000); //Sending the currently running thread into Sleeping State
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            else {
                System.out.println("Welcome");
                try {
                    Thread.sleep(500); //Sending the currently running thread into Sleeping State
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
}
```

```
package multithreading;
//This is the Thread implementation class
public class GreetingRunnableUsingCurrentThread implements Runnable{

    public void run() {
        for(int a=1;a<=10;a++) {
            Thread currThread = Thread.currentThread();
            String name = currThread.getName();
            if(name.equals("first")) {
                System.out.println("Hello");
                try {
                    Thread.sleep(1000); //Sending the currently running thread into Sleeping State
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            else {
                System.out.println("Welcome");
                try {
                    Thread.sleep(500); //Sending the currently running thread into Sleeping State
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
}
```

```
1 package multithreading;
2
3 public class GreetingRunnableUsingCurrentThreadExample {
4
5     public static void main(String[] args) {
6         Runnable r1 = new GreetingRunnableUsingCurrentThread();
7         Runnable r2 = new GreetingRunnableUsingCurrentThread();
8         Thread t1 = new Thread(r1, "first");
9         Thread t2 = new Thread(r2, "second");
10        t1.start();
11        t2.start();
12        try {
13            //Call to join() causes the parent thread: main thread to wait until the death of the threads
14            //on which it is invoked
15            t1.join();
16            t2.join();
17        } catch (InterruptedException e) {
18            // TODO Auto-generated catch block
19            e.printStackTrace();
20        }
21
22        System.out.println("THANK YOU");
23    }
24
25
26
27 }
```

GreetingRunnableUsingCurrentThread.java X GreetingRunnableUsingCurrentThreadExample.java X SynchronizationExample.java X Message.java X MessageThread.java X StorageExample.java X StorageThread.java X Multithreading Ass X

```
1 package multithreading;
2
3 public class SynchronizationExample {
4
5     public static void main(String[] args) {
6         Message msg = new Message("Hello Everyone !!");
7         Thread t1 = new MessageThread(msg);
8         Thread t2 = new MessageThread(msg);
9         t1.start();
10        t2.start();
11    }
12
13 }
14
15 }
```

GreetingRunnableUsingCurrentThread.java GreetingRunnableUsingCurrentThreadExample.java SynchronizationExample.java Message.java MessageThread.java StorageExample.java StorageThread.java Multithreading Ass

```
1 package multithreading;
2
3 public class Message {
4     private String content;
5
6     public Message(String content) {
7         this.content = content;
8     }
9     public void printMessage() throws InterruptedException{
10        System.out.println("*****");
11        Thread.sleep(1000);
12        System.out.println(content);
13        Thread.sleep(1000);
14        System.out.println("*****");
15        Thread.sleep(1000);
16    }
17
18
19
20
21 }
```

GreetingRunnableUsingCurrentThread.java GreetingRunnableUsingCurrentThreadExample.java SynchronizationExample.java Message.java MessageThread.java StorageExample.java StorageThread.java Multithreading As

```
1 package multithreading;
2
3 public class MessageThread extends Thread {
4     private Message messageObject;
5
6     public MessageThread(Message messageObject) {
7         this.messageObject = messageObject;
8     }
9
10    public void run() {
11        synchronized (messageObject) {
12            try {
13                messageObject.printMessage();
14            } catch (InterruptedException e) {
15                // TODO Auto-generated catch block
16                e.printStackTrace();
17            }
18        }
19    }
20
21 }
22
23 }
```

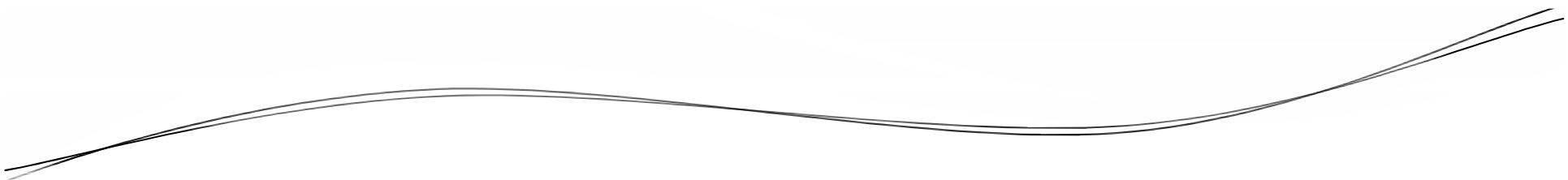
GreetingRunnableUsingCurrentThread.java GreetingRunnableUsingCurrentThreadExample.java SynchronizationExample.java Message.java Message Thread.java StorageExample.java Storage Thread.java Multithreading Ass

```
1 package example.multithreading.interthread.communication;
2
3 public class StorageExample {
4
5     public static void main(String[] args) {
6         Storage str = new Storage(); //valueSet: false, num: 0
7         Thread t1 = new StorageThread(str, "Producer");
8         Thread t2 = new StorageThread(str, "Consumer");
9         t1.start();
10        t2.start();
11    }
12
13 }
14
15 //if(!valueSet)
16 //if(valueSet)
17
18
19
```

```
1 package example.multithreading.interthread.communication;
2
3 public class StorageThread extends Thread {
4     private Storage dataStorage;
5
6     public StorageThread(Storage dataStorage, String name) {
7         super(name);
8         this.dataStorage = dataStorage;
9     }
10    public void run() {
11        synchronized (dataStorage) {
12            for(int a=1;a<=5;a++) {
13                Thread currThread = Thread.currentThread();
14                String tName = currThread.getName();
15                if(tName.equals("Producer")) {
16                    dataStorage.setNum(a);
17                }
18                else if(tName.equals("Consumer")) {
19                    int square = dataStorage.getSquare();
20                    System.out.println("Square: " + square);
21                    try {
22                        Thread.sleep(1000);
23                    } catch (InterruptedException e) {
24                        // TODO Auto-generated catch block
25                        e.printStackTrace();
26                    }
27                }
28            }
29        }
30    }
31 }
32 }
```

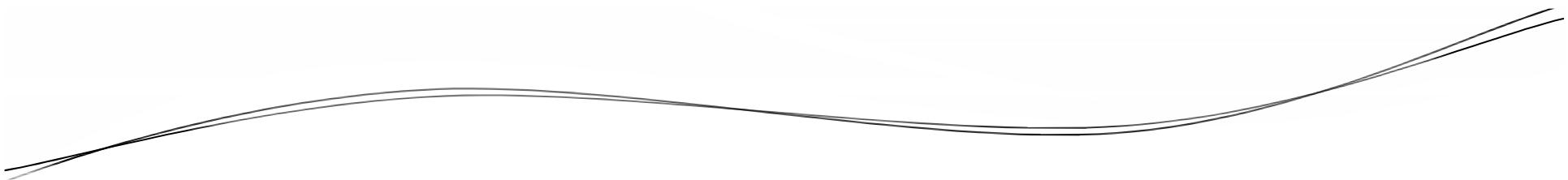
```
1 1. Write a program that creates 2 threads to perform following
2 operations:
3 The first thread prints all the English alphabets
4 in Uppercase
5
6 The second thread prints the values of the alphabets
7
8 E.g.
9 First Thread: A, B, C and so on
10 Second Thread: 65, 66, 67 and so on
11 Both the threads must print their values using a same
12 time gap.
13
14 2. Create a class PaytmAccount with following attributes:
15     mobileNo (String)
16     name (String)
17     balance (int)
18
19     Implement 2 additional methods:
20         void addMoney(int)
21         void makePayment(int)
22
23 Write a program that creates 2 threads in such a way that both
24 the threads use the same object of PaytmAccount and one thread
25 adds the money whereas the another thread makes the payment.
26
27 Display messages with current balance using some time gap
28 while performing both the transactions.
29
30 Note***  

31 (When one thread is using the object, access should be denied
32 for the other thread.)
```



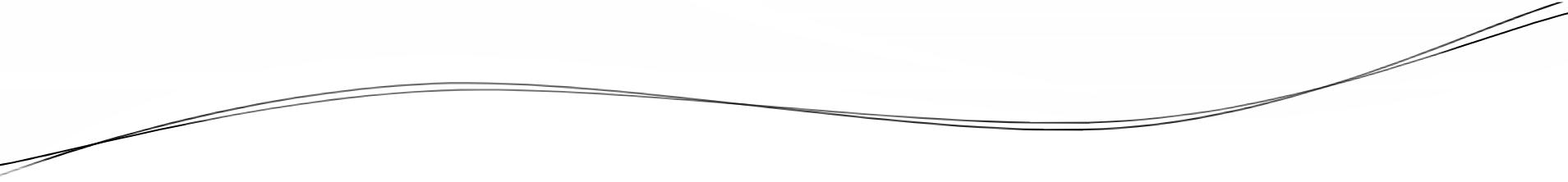
IO Programming

By Rahul Barve



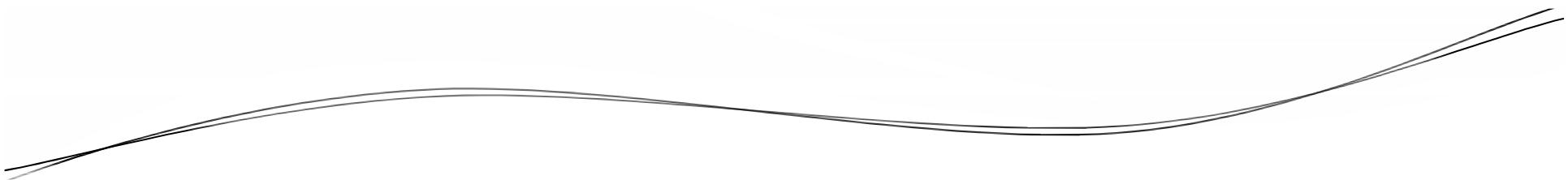
Objectives

- Introduction to Streams.
- Understanding I/O Hierarchy.
- Stream Types.
- Understanding Serialization
- NIO Basics



Introduction to Streams

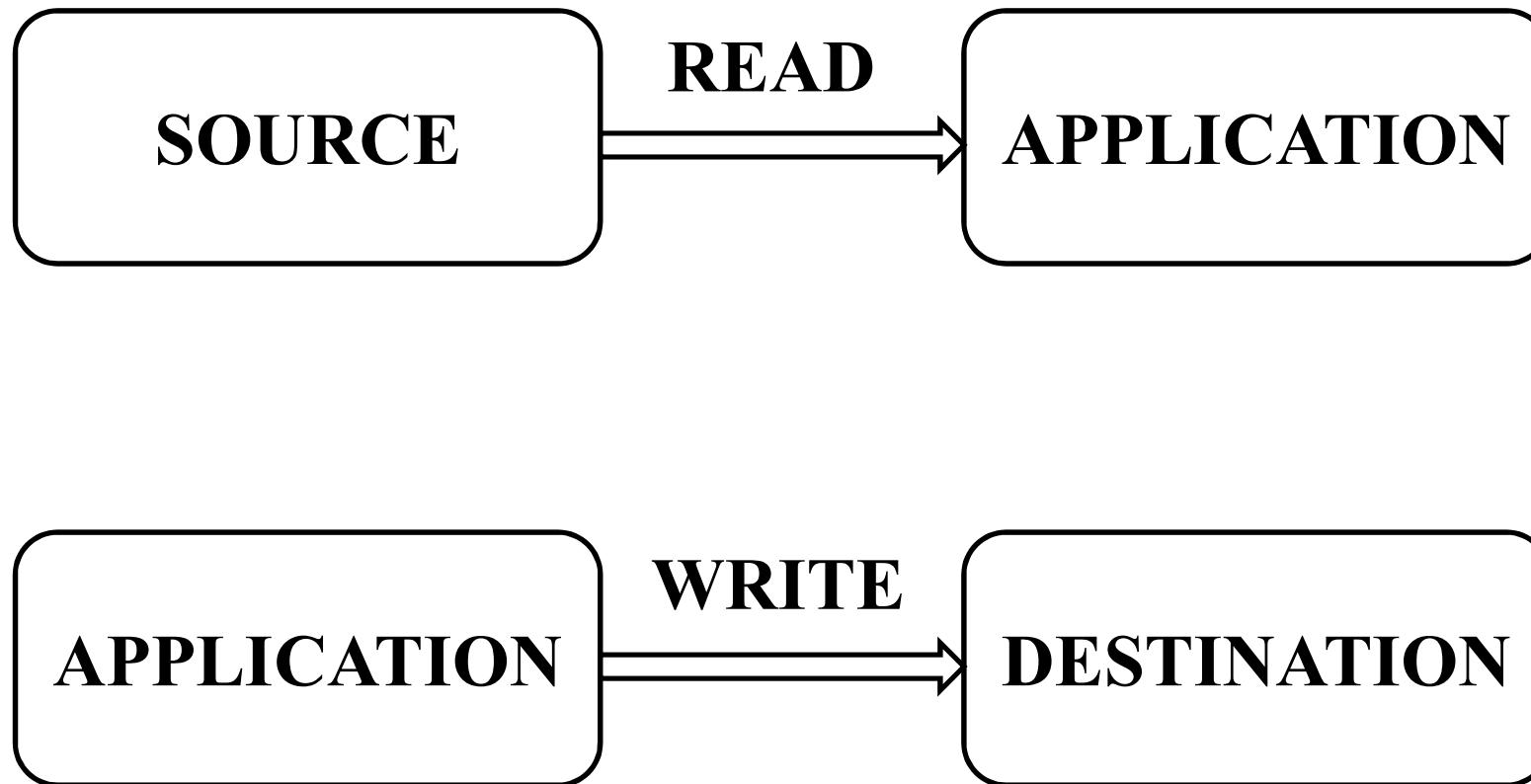
By Rahul Barve

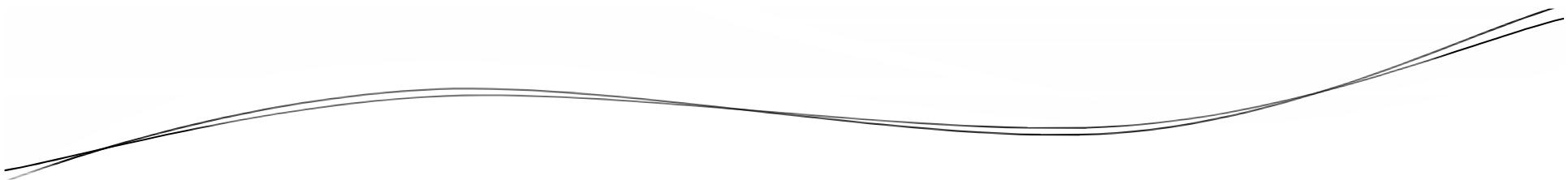


Introduction to Streams

- Every application needs to read and write the data.
- To read the data, application requires some source and to write into, it requires some destination.

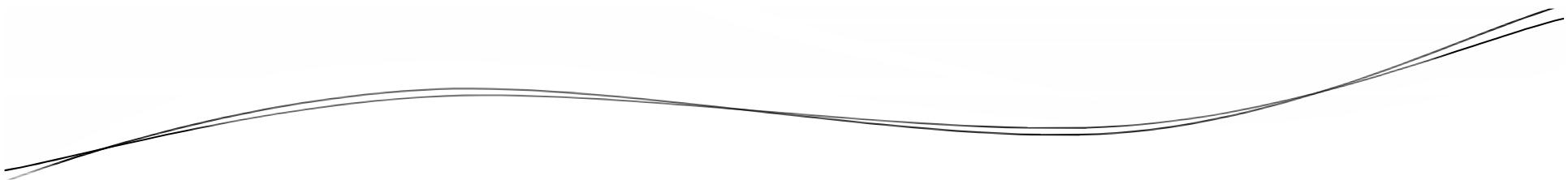
Introduction to Streams





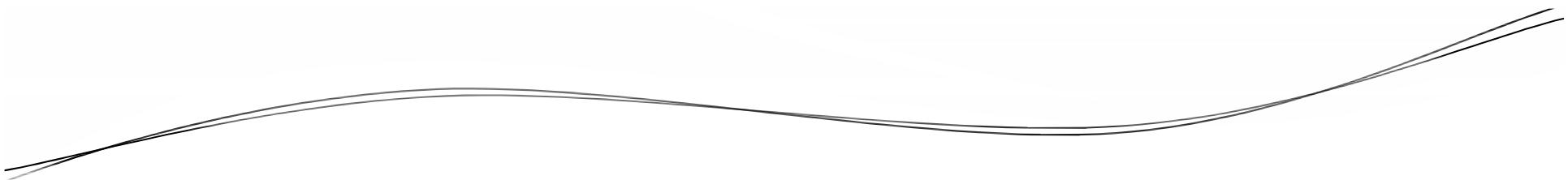
Introduction to Streams

- An application can read the data from several sources like File, Input Device, Socket and so on.
- Similarly it can write the data into some destination like File, Output Device, Socket and so on.



Introduction to Streams

- It is possible by having a connector in between the application and the source or the destination.
- This connector is known as a stream.

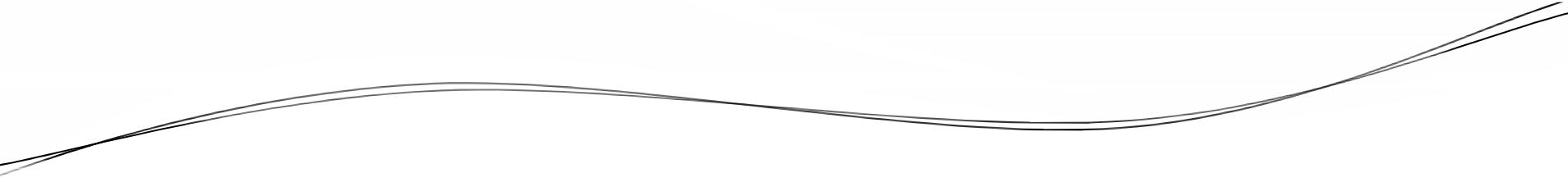


Introduction to Streams

- At the base, Java library provides two types of streams: `InputStream` and `OutputStream`.
- Java provides IO library support using a package `java.io`.

Introduction to Streams

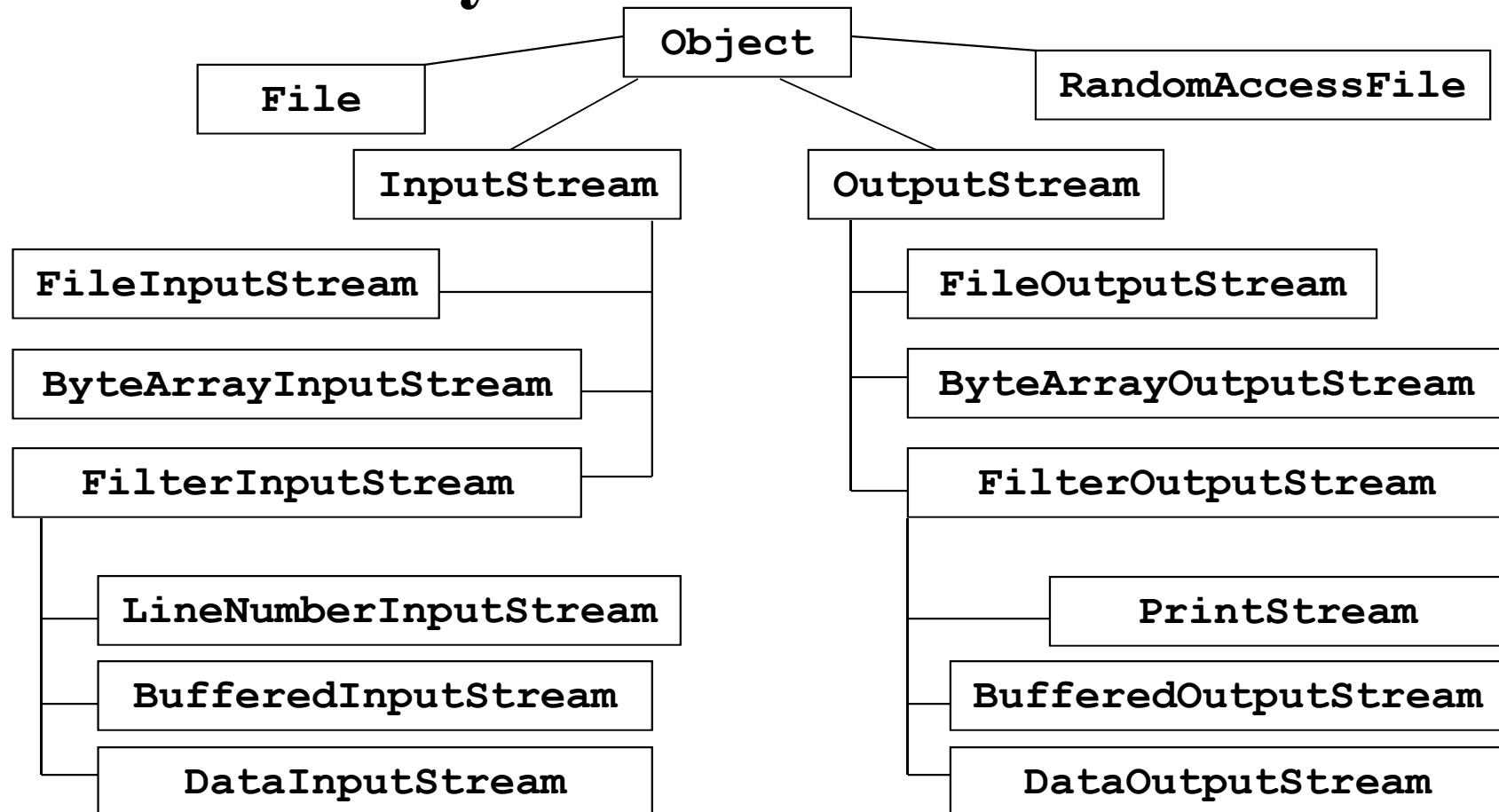
- `InputStream` – This depicts the flow of bytes from data source to the program's memory.
- `OutputStream` – This depicts the flow of bytes from program's memory to the destination data store.

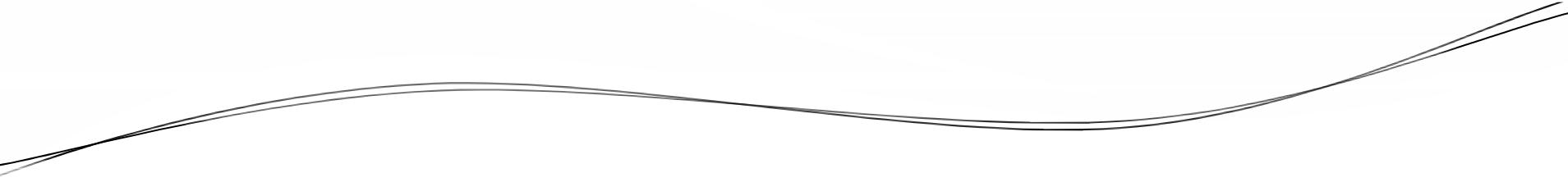


I/O Hierarchy

By Rahul Barve

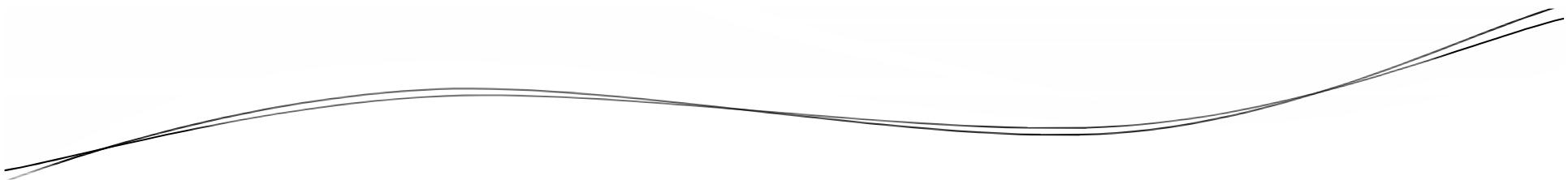
I/O Hierarchy





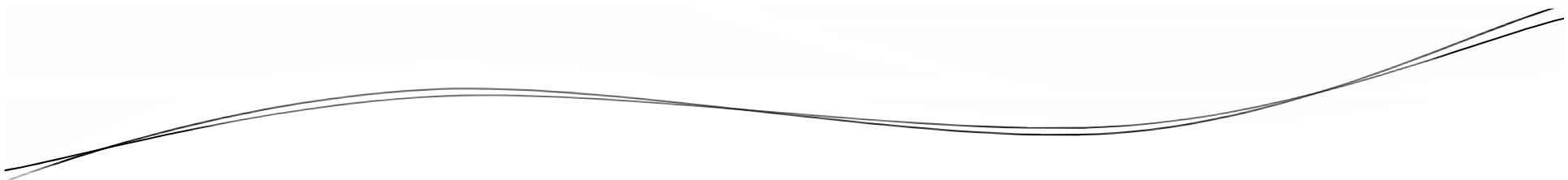
I/O Hierarchies

By Rahul Barve



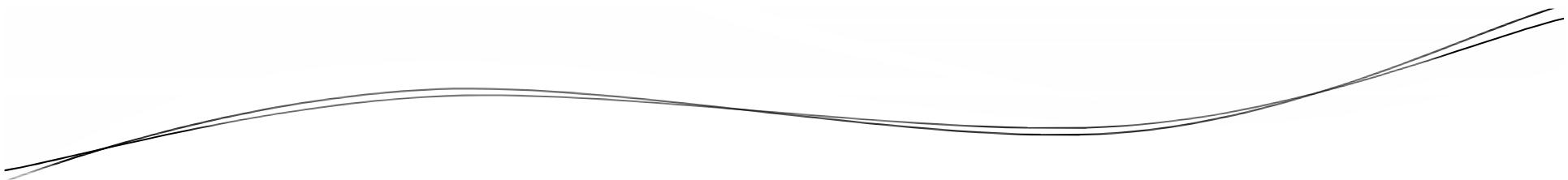
I/O Hierarchies

- There are 2 types of I/O Hierarchies available in Java
 - Byte Streams
 - Character Streams



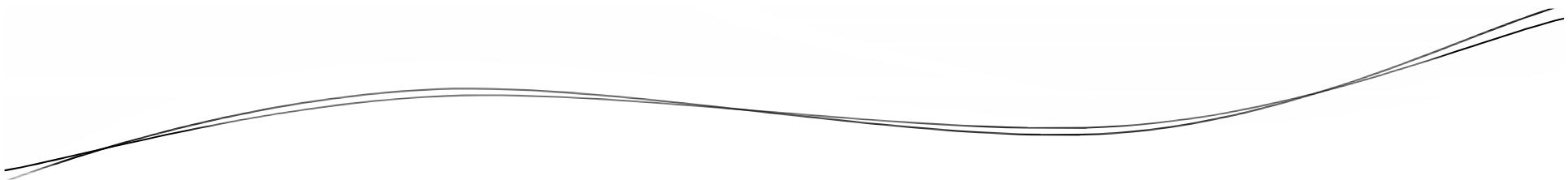
I/O Hierarchies

- Byte streams work upon 8 bit data and fall under `InputStream` and `OutputStream` hierarchies.
- Character streams work upon 16 bit data and fall under `Reader` and `Writer` hierarchies.



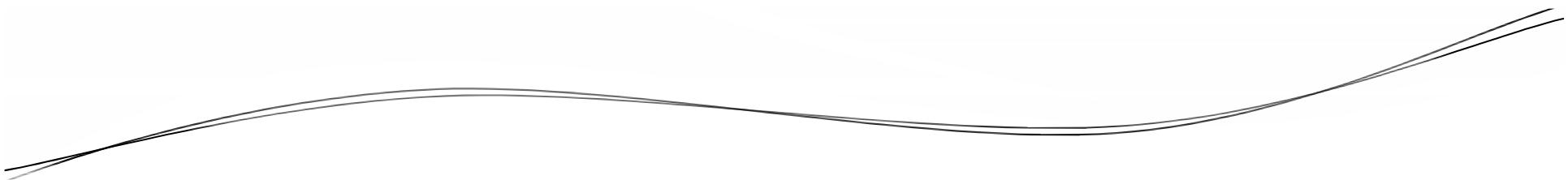
InputStream

By Rahul Barve



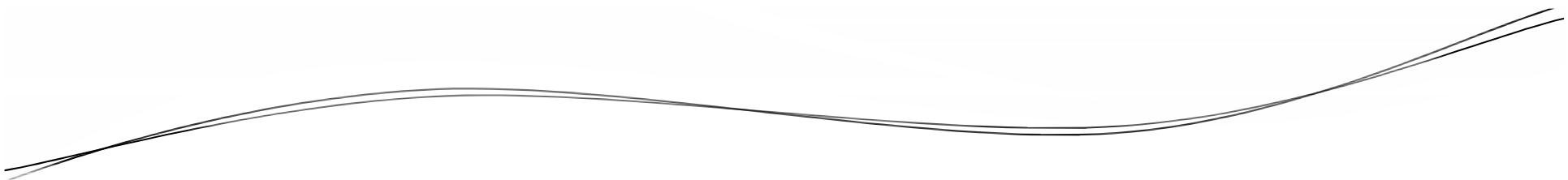
InputStream

- An abstract base class that defines methods for performing READ operations.
- Defines a basic interface for reading streamed bytes of information.



FileInputStream

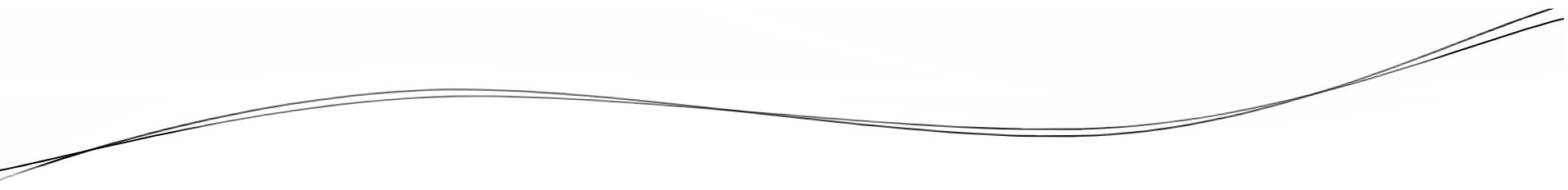
By Rahul Barve



FileInputStream

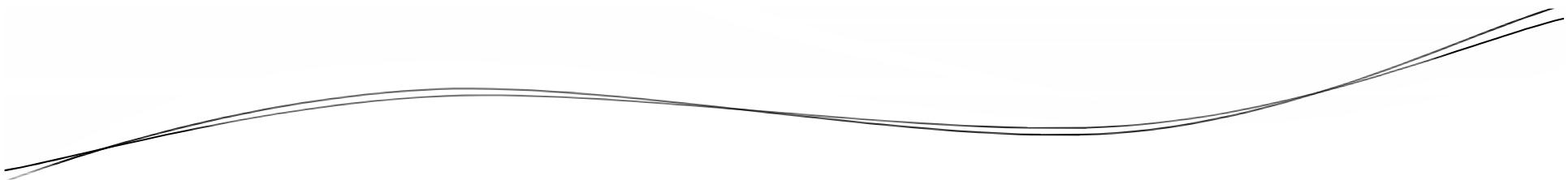
- Used to read contents of a file.
- E.g.

```
FileInputStream fin;  
String file = "message.txt";  
fin = new FileInputStream(file);
```



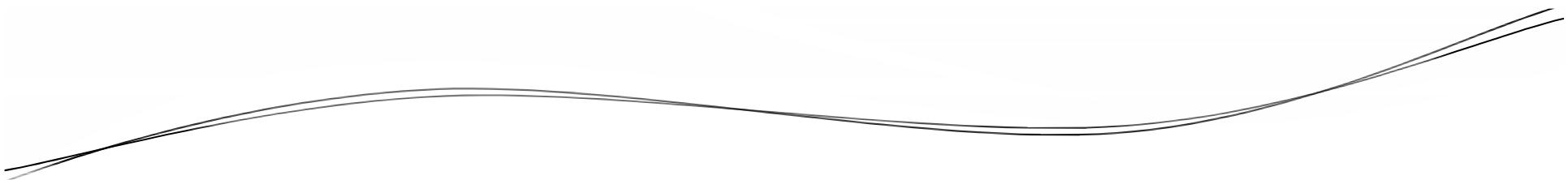
try-with-resources

By Rahul Barve



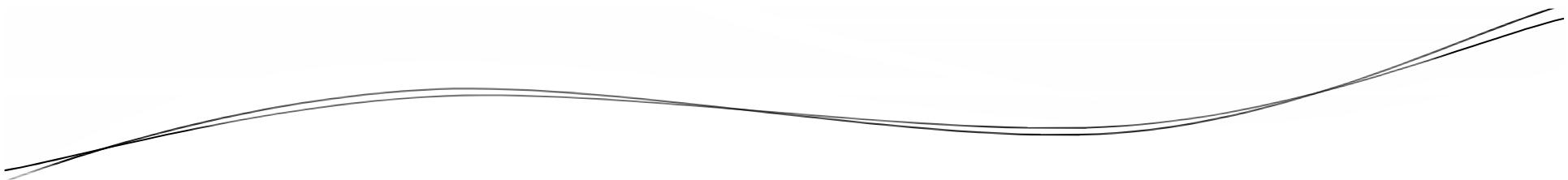
try-with-resources

- A feature introduced since Java 1.7.
- A `try` statement that declares one or more resources.



try-with-resources

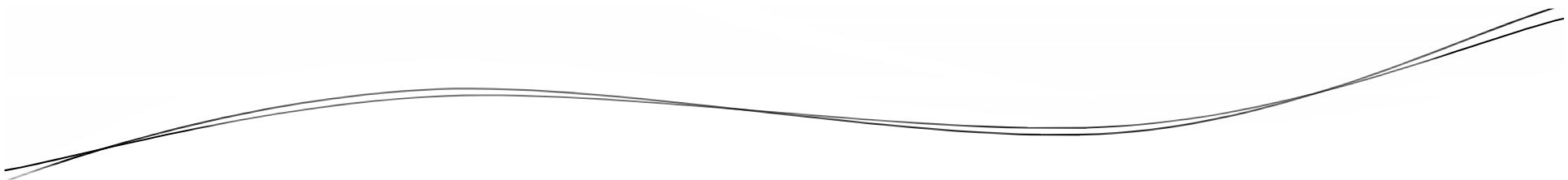
- A resource is an object that must be closed after the program is finished with it.
- Any object that implements `java.lang.AutoCloseable` can be used as a resource.



try-with-resources

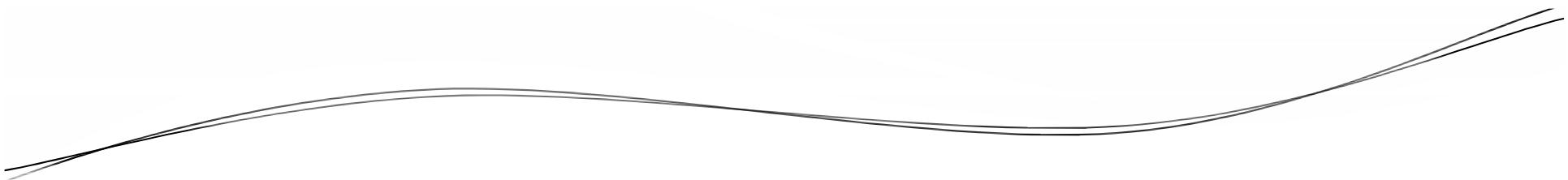
- E.g.

```
try (FileInputStream fin = ....) {  
    //Statements  
}  
  
catch (...) {  
    //Statements  
}
```



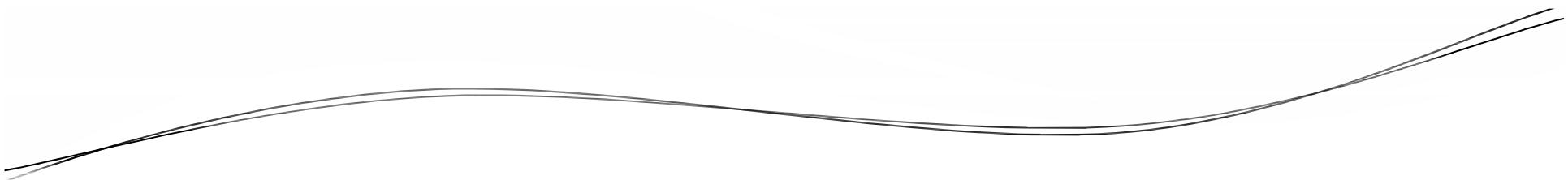
OutputStream

By Rahul Barve



OutputStream

- An abstract base class that defines methods for performing WRITE operations.
- Defines a basic interface for writing streamed bytes.



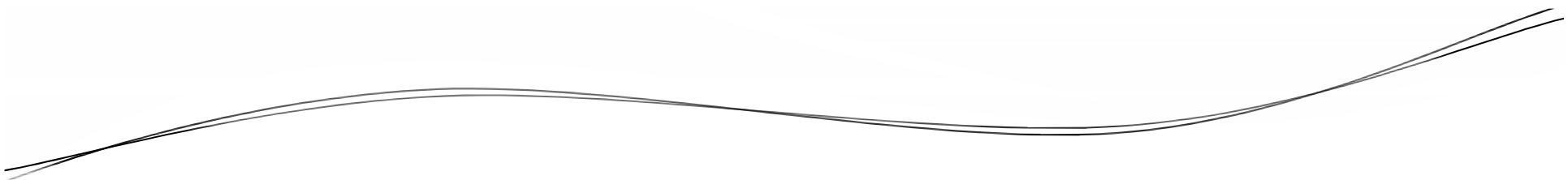
FileOutputStream

By Rahul Barve

FileOutputStream

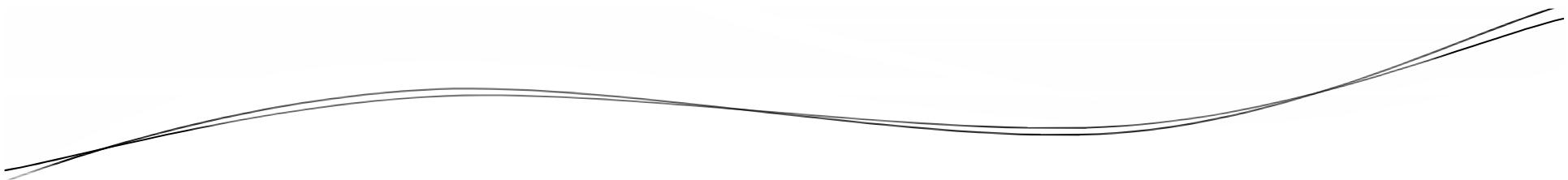
- Used to write the contents to a file.
- E.g.

```
String file = "message.txt";  
FileOutputStream fout;  
fout = new FileOutputStream(file);
```



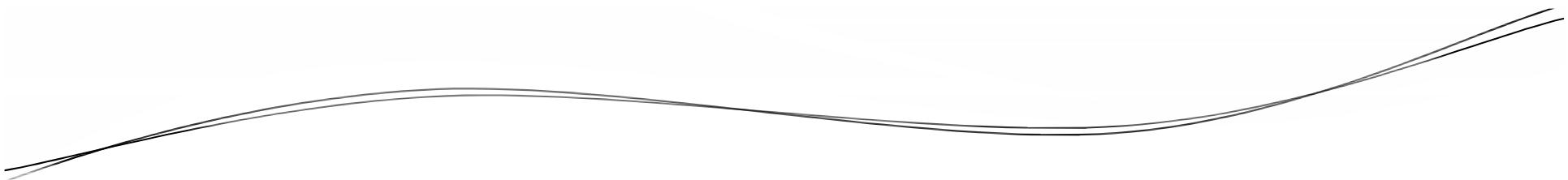
FileOutputStream

- Use 2 parameterized constructor with 2nd parameter as boolean true to open the file in append mode.



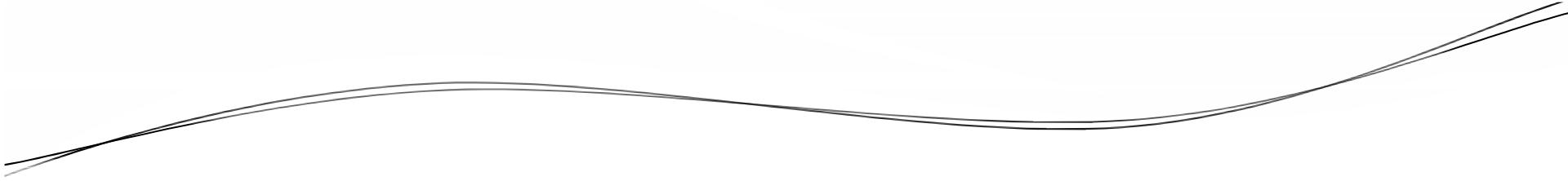
FilterInputStream

By Rahul Barve



FilterInputStream

- A base class for all other classes that act like a filter to transform the raw bytes to a desired form.
- Must be used through one of the sub classes.

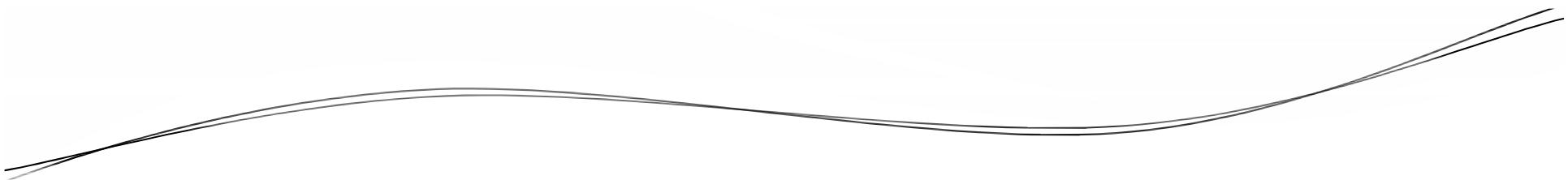


FilterInputStream Subclasses

- SequenceInputStream
- LineNumberInputStream
- BufferedInputStream
- DataInputStream

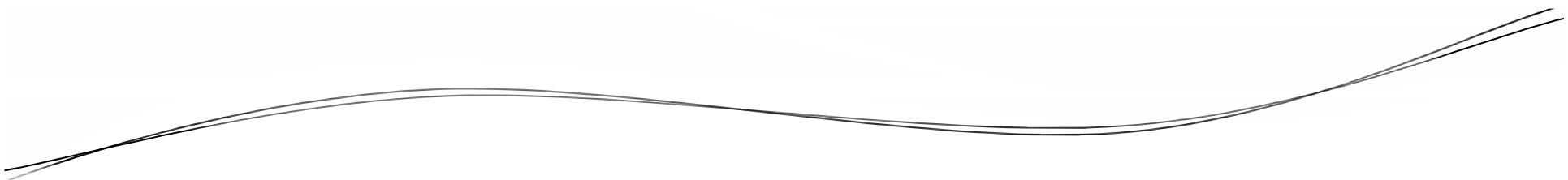
FilterInputStream Subclasses

- SequenceInputStream
 - Used to represent a sequential input stream.
- LineNumberInputStream
 - Used to operate on the basis of line numbers.
- BufferedInputStream
 - Used to populate an input stream on the top of buffer.
 - Improves performance.
- DataInputStream
 - Used to read Java's basic primitive data types.



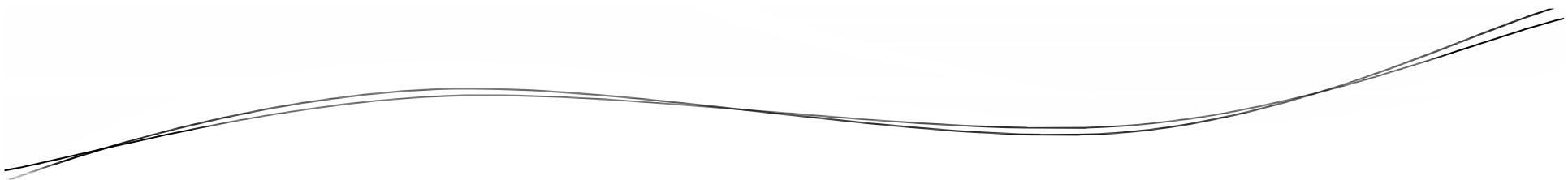
File Class

- Used to access information about some specific file.
- Objects of File do not actually open any file.



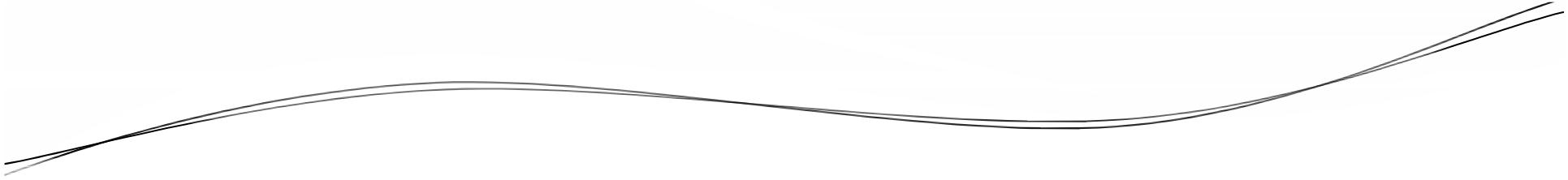
File Class

- Methods:
 - boolean exists()
 - boolean isDirectory()
 - boolean isFile()
 - boolean createNewFile()
 - boolean delete()
 - boolean mkdir()



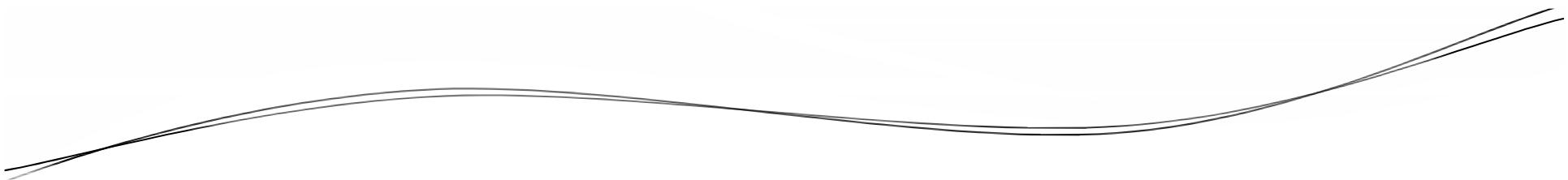
RandomAccessFile

By Rahul Barve



RandomAccessFile

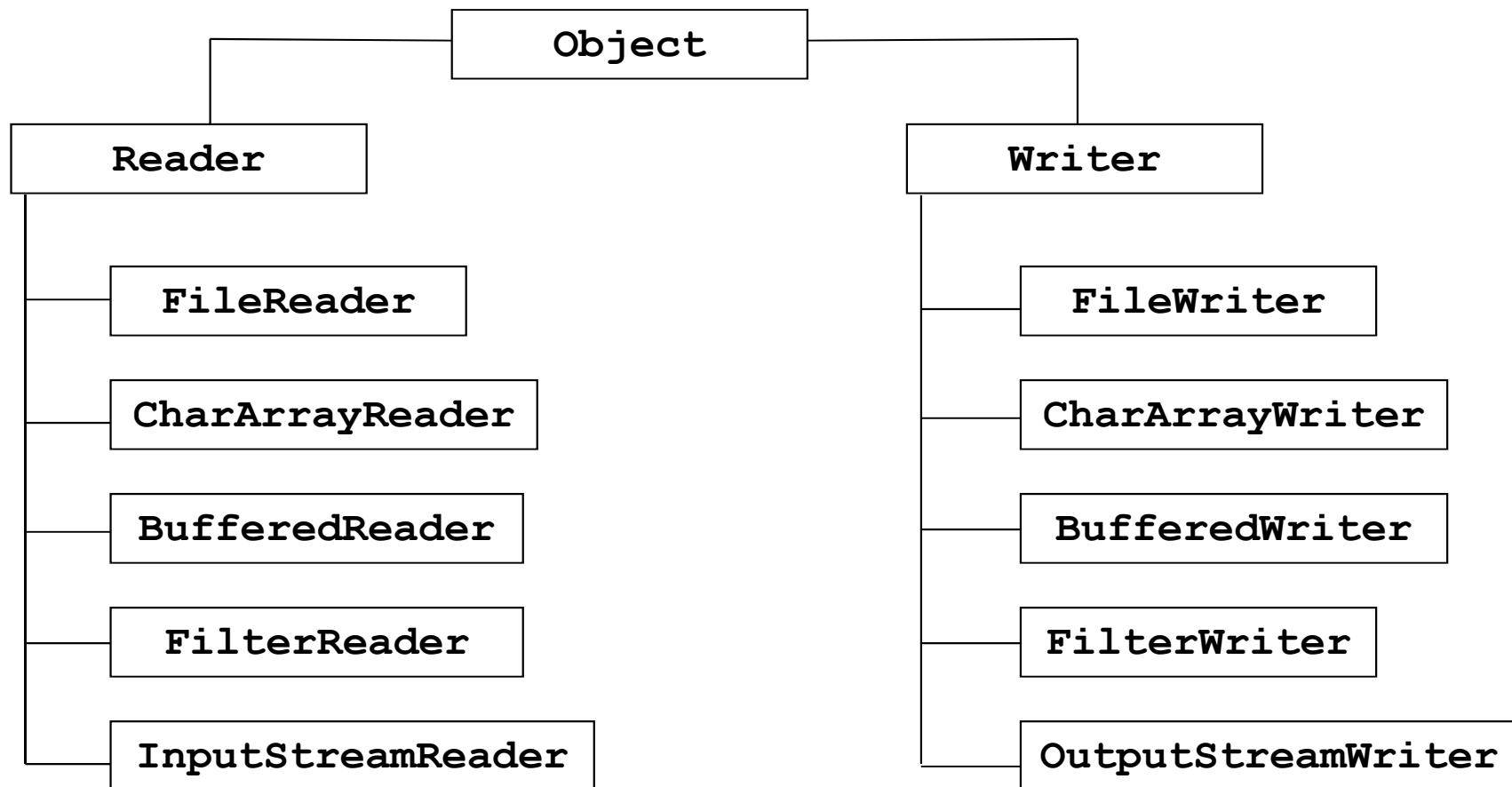
- Allows to place file pointer anywhere in the file, using `seek()` method.
- Can be used to perform both operations:
 - Read
 - Write
- Inherited from `Object` class directly.

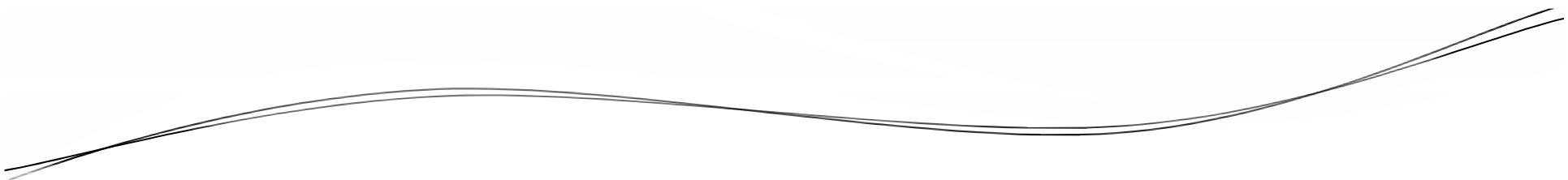


Reader – Writer Hierarchy

By Rahul Barve

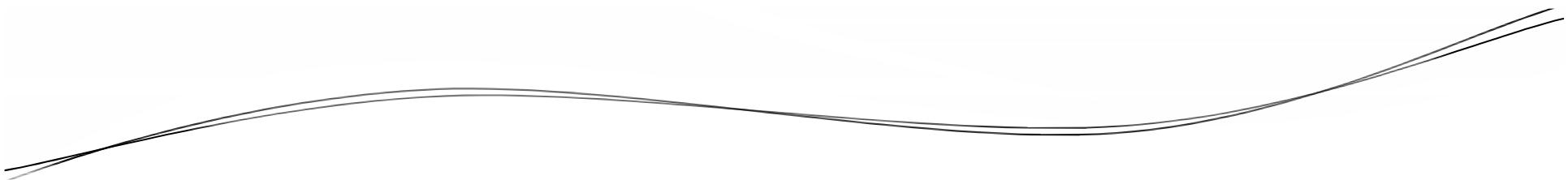
Reader – Writer Hierarchy





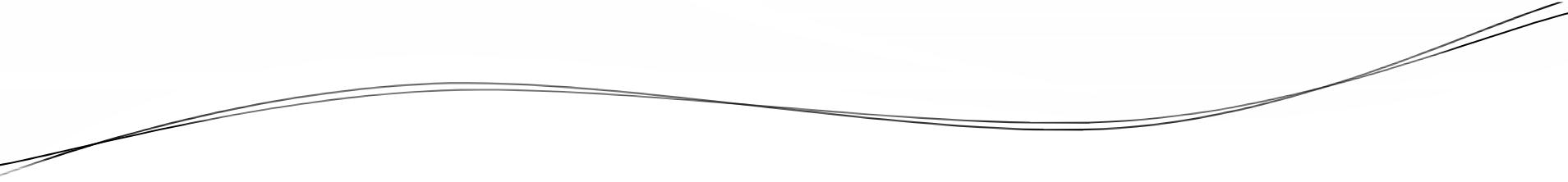
BufferedReader

By Rahul Barve



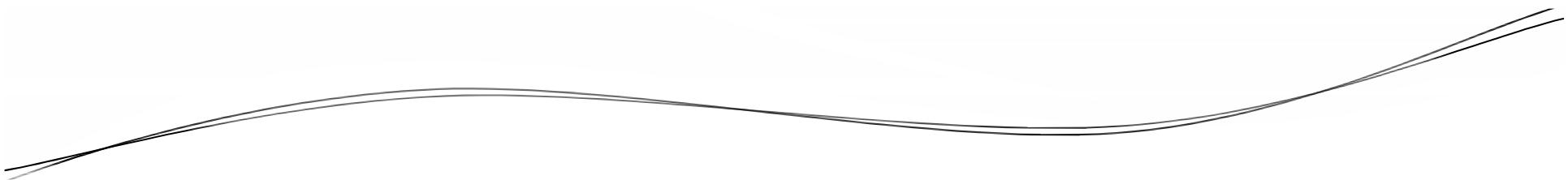
BufferedReader

- A character stream used to apply a buffering model for performance optimization.
- Provides a method `readLine()` which allows to read the contents line by line.



System Class

By Rahul Barve



System Class

- A class from `java.lang` package.
- Provides public static final members to read from input device as keyboard and write to the console of the system.

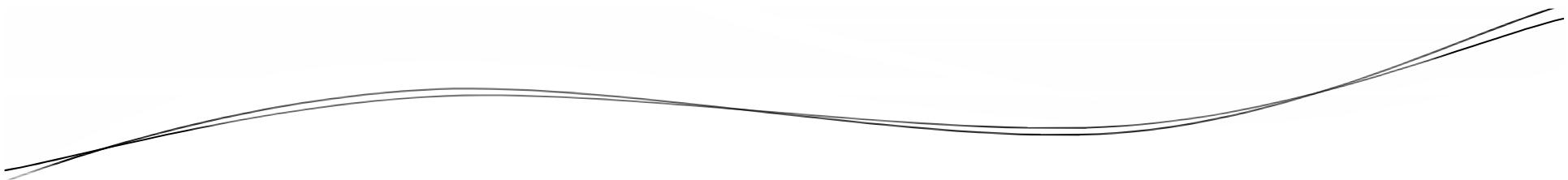
System Class

- `System.in`

Reference of type `java.io.InputStream` that refers to the I/P device i.e. Keyboard.

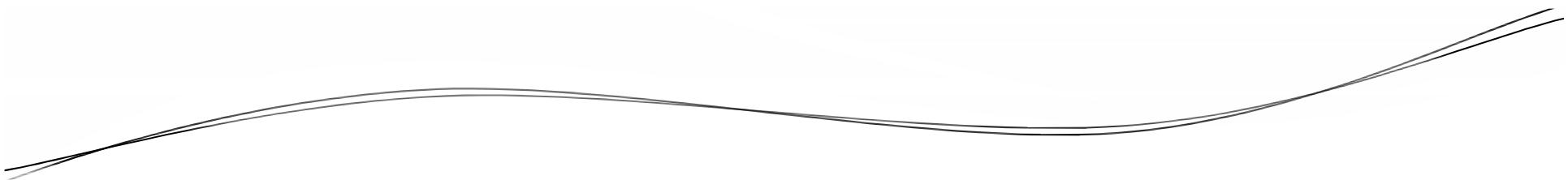
- `System.out`

Reference of type `java.io.PrintStream` that refers to the O/P device i.e. Console.



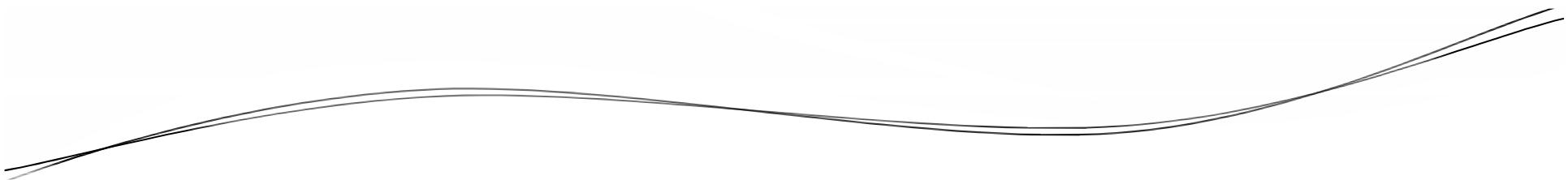
Serialization

By Rahul Barve



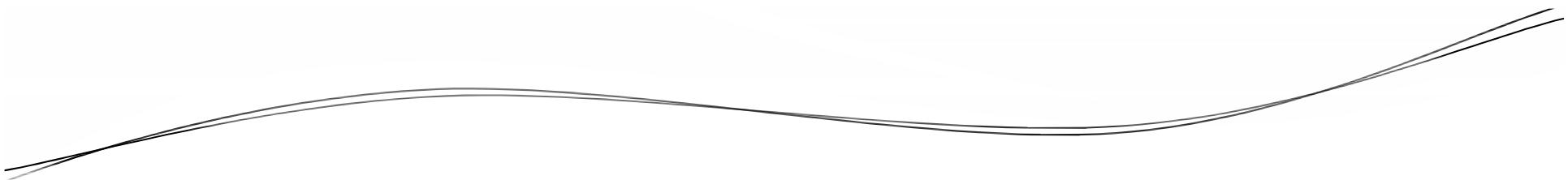
Serialization

- The process of storing the state of an object to some permanent persistent store is called as Serialization.



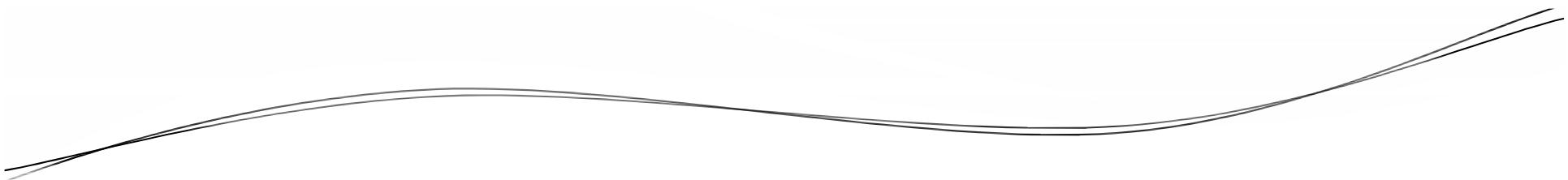
Serialization API

- To perform Serialization and De-Serialization, Java provides a relevant API known as Serialization API.



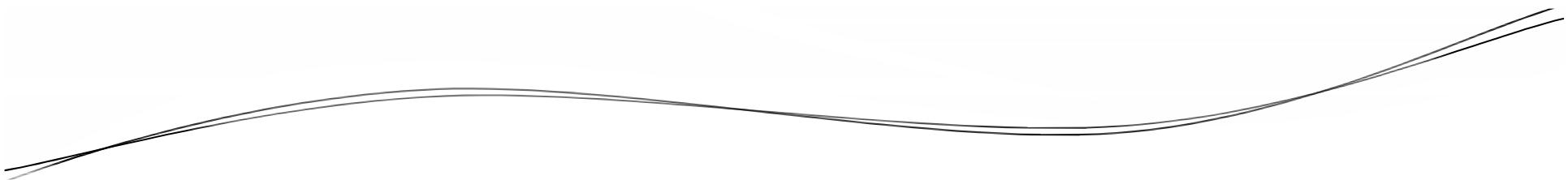
Serialization API

- Serializable
- ObjectOutputStream
- ObjectInputStream



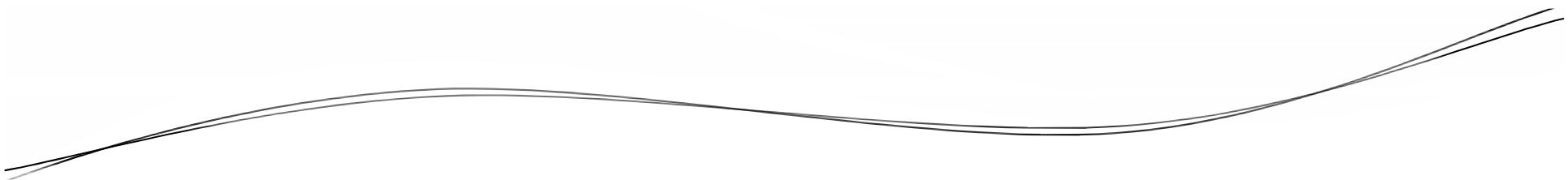
Serializable

By Rahul Barve



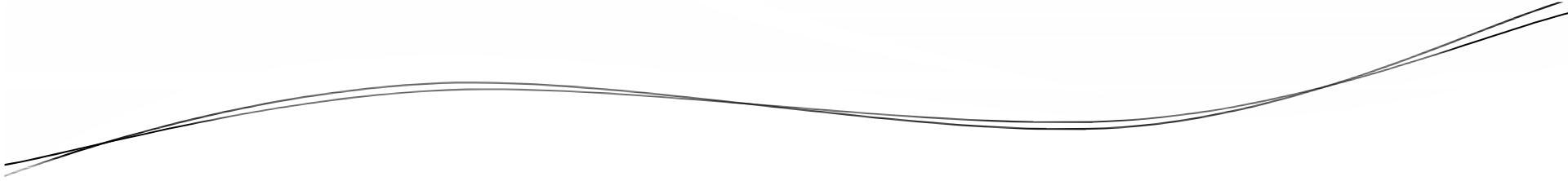
Serializable

- A marker interface
- Needs to be implemented by a class of which an object is to be serialized.



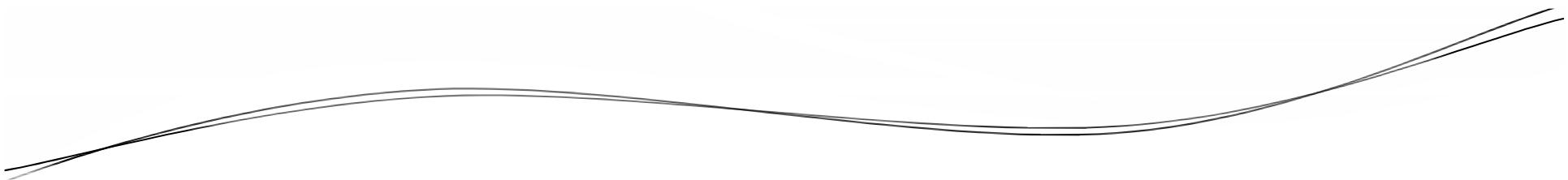
ObjectOutputStream

By Rahul Barve



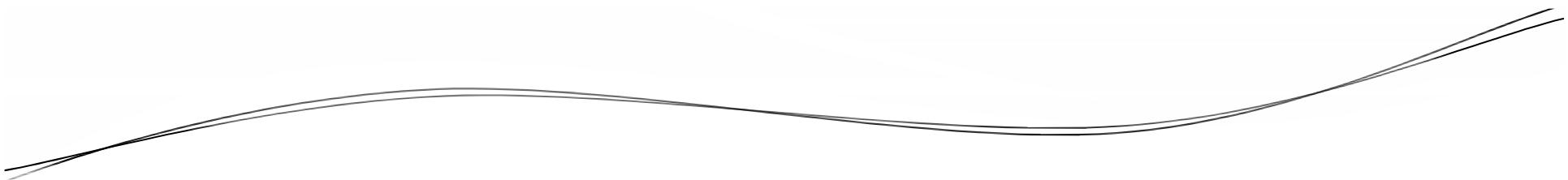
ObjectOutputStream

- A class used to perform serialization.
- Uses OutputStream as a target and provides void writeObject(Object) method to perform the serialization.



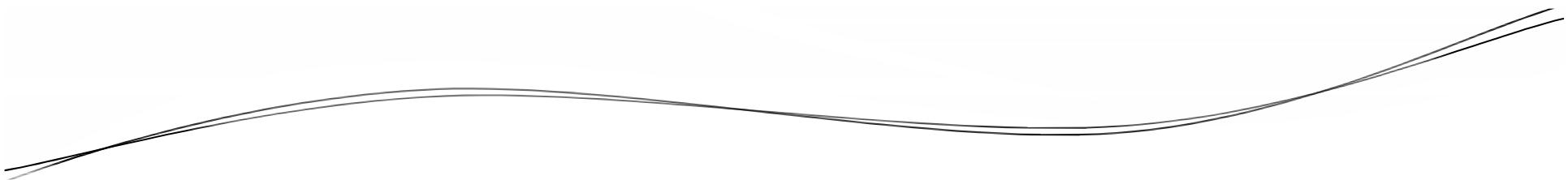
ObjectInputStream

By Rahul Barve



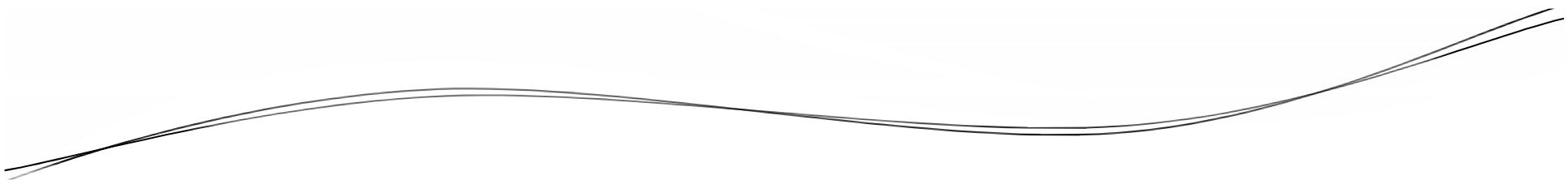
ObjectInputStream

- A class used to perform de-serialization.
- Uses InputStream as a source and provides Object readObject () method to perform the de-serialization.



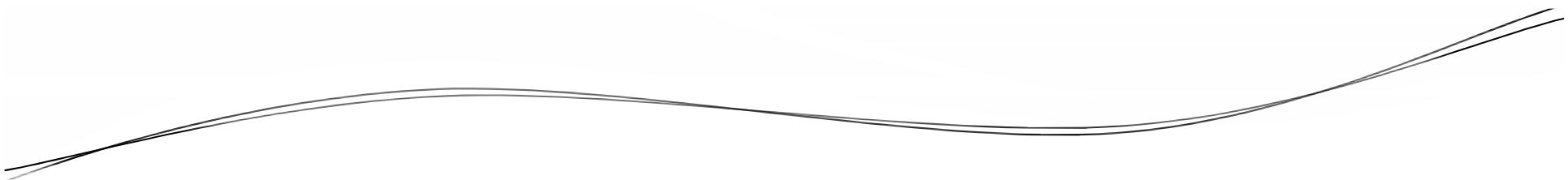
Externalizable

By Rahul Barve



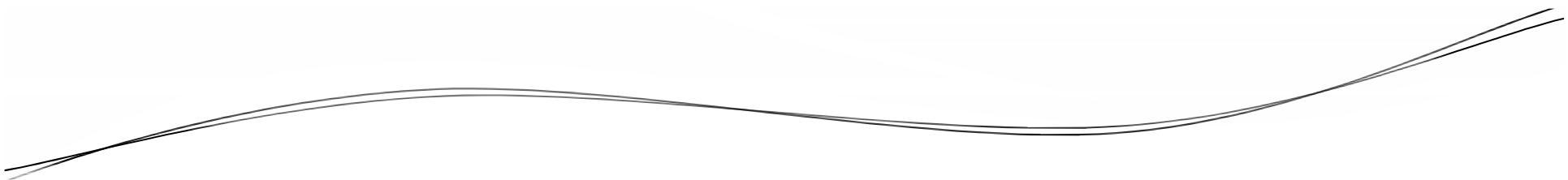
Externalizable

- A sub-interface of Serializable.
- Allows to have full control over encryption and decryption while performing serialization and de-serialization respectively.



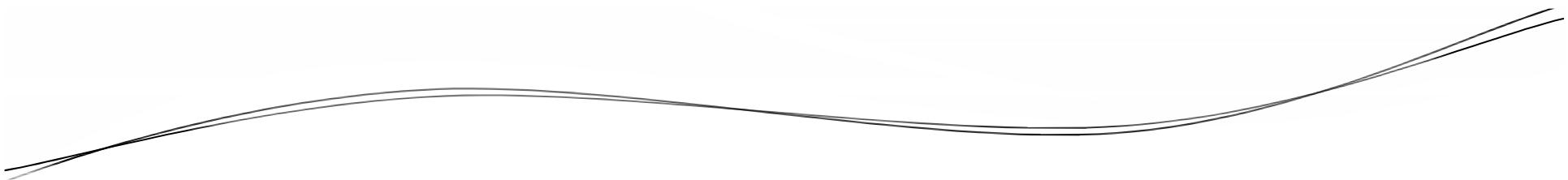
Externalizable

- Methods:
 - `readExternal(ObjectInput);`
 - `writeExternal(ObjectOutput);`



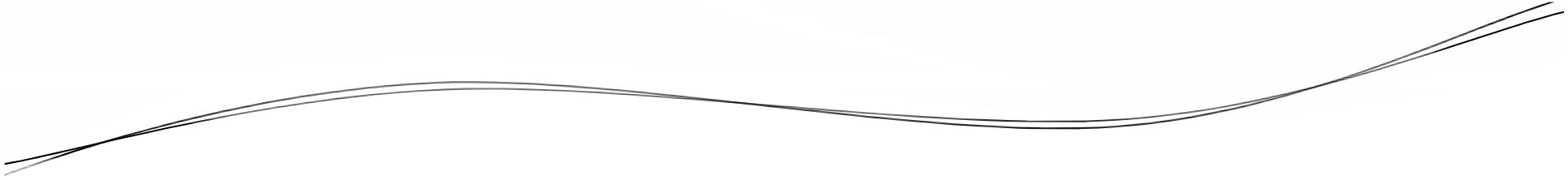
Serialization Further Concepts

By Rahul Barve



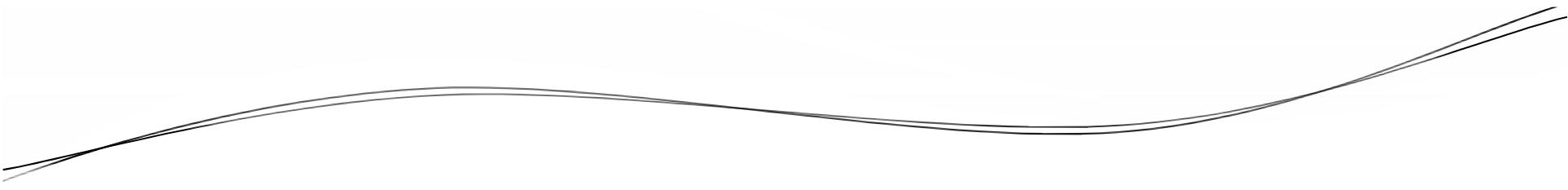
Serialization Further Concepts

- If a variable of a class is declared as static, its value does not get serialized.
- It gets initialized with default values during de-serialization.



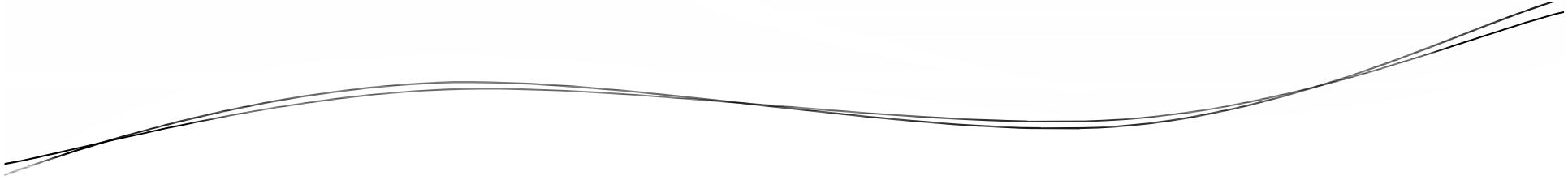
Serialization Further Concepts

- In order to avoid serialization for an instance variable, the instance variable is to be declared as transient.
- Transient variables are also initialized with default values during de-serialization.



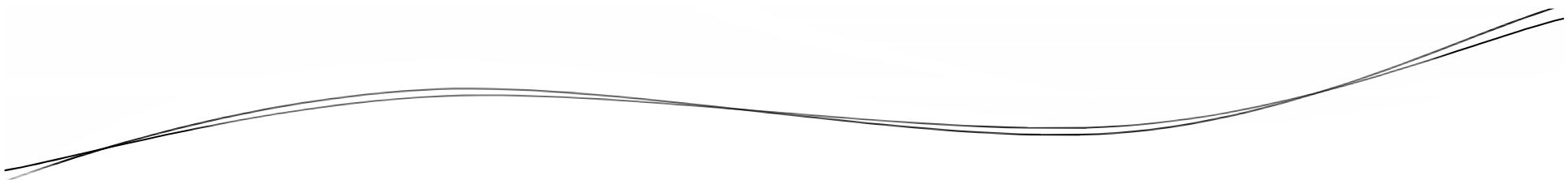
NIO Basics

By Rahul Barve



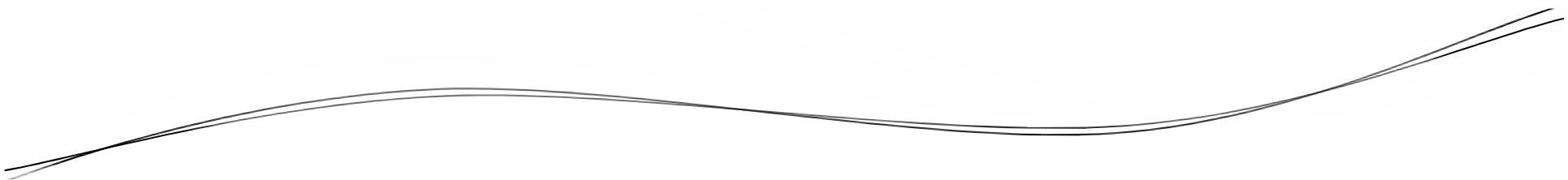
NIO Basics

- NIO stands for New IO
- A separate API provided by Java for Performance Optimization



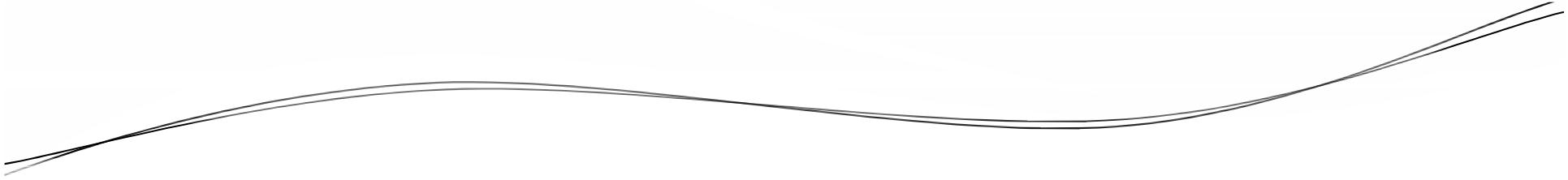
NIO Basics

- The NIO specific library mainly consists of 3 packages:
 - `java.nio`
 - `java.nio.file`
 - `java.nio.channels`



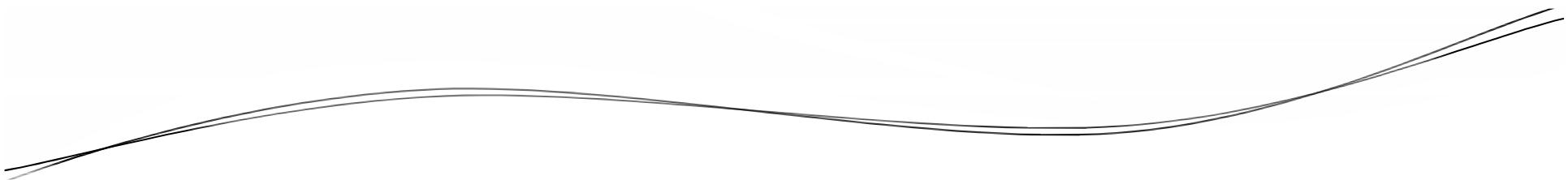
NIO Basics

- The `java.nio` package provides several types of *buffers* which are used as containers for the data.
- E.g. `ByteBuffer`



NIO Basics

- The `java.nio.channels` package provides several types of *channels* which are used to transfer the data from *source* to *buffer* or from *buffer* to some *destination*.
- E.g. `FileChannel`



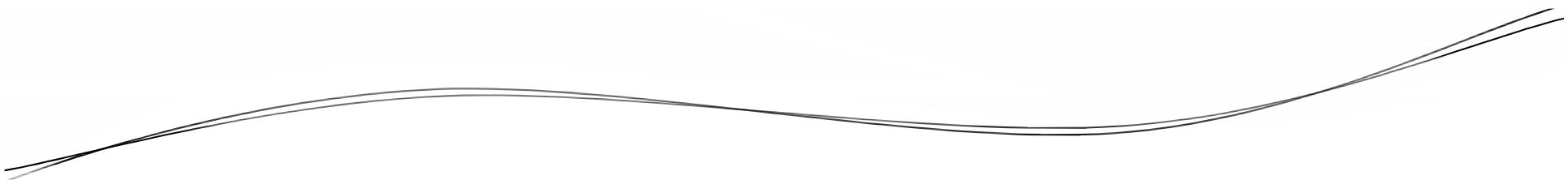
NIO Basics

- The `java.nio.file` package provides several utilities to handle file operations.
- E.g.
 - Path
 - Paths
 - Files

NIO Basics

- The interface named as Path indicates the path of the resource and it can be obtained using Paths class.
- E.g.

```
Path currentPath =  
    Paths.get("----somepath---");
```



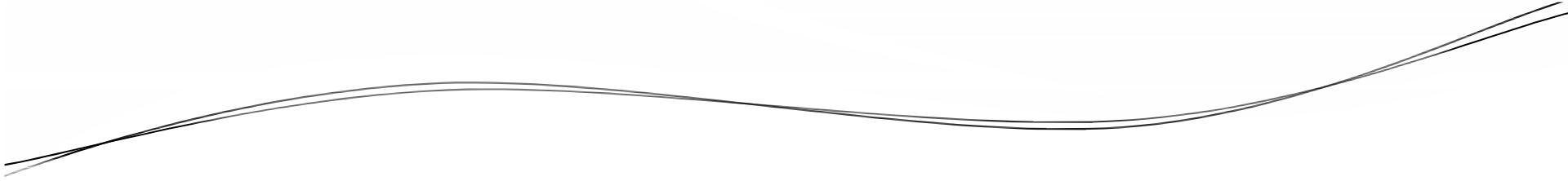
NIO Basics

- The class Files is used to handle file specific operations.
- It provides several utility methods
- E.g.

readAllLines()

write()

exists()



Lets Summarize

- What are Streams
- Types of Streams
- Working with Input and Output
- Reader and Writer
- Object Serialization
- NIO Basics

```
FileReadExampleUsingTryWithResources.java X FileWriteExample.java X FileReadExampleUsingBufferedInputStream.java X RandomAccessFileExaple.java X BufferedReaderExample.java X UserInputExample.java X
```

```
1 package io_programming;
2
3 import java.io.FileInputStream;
4 import java.io.FileNotFoundException;
5 import java.io.IOException;
6 import java.io.InputStream;
7
8 public class FileReadExampleUsingTryWithResources {
9
10    public static void main(String[] args) {
11        // TODO Auto-generated method stub
12        String filePath = "./src/resources/technologies.txt";
13
14        try(FileInputStream in = new FileInputStream(filePath)) {
15            //Resource is initialized and instantiated
16            while(true) {
17                //Read the content character by character using read()
18                int data = in.read(); //Resource is getting used
19                //The read() method reads 1 character and returns its value and shifts the pointer 1 step ahead
20                //When EOF is encountered, the method returns -1
21                if(data == -1)
22                    break;
23                char ch = (char)data;
24                System.out.print(ch);
25            }
26        } catch (FileNotFoundException e) {
27            // TODO Auto-generated catch block
28            e.printStackTrace();
29        } catch (IOException e) {
30            // TODO Auto-generated catch block
31            e.printStackTrace();
32        }
33    }
34}
```

FileReadExampleUsingTryWithResources.java | FileWriteExample.java | FileReadExampleUsingBufferedInputStream.java | RandomAccessFileExaple.java | BufferedReaderExample.java | UserInputExample.java

```
25     }
26 } catch (FileNotFoundException e) {
27     // TODO Auto-generated catch block
28     e.printStackTrace();
29 } catch (IOException e) {
30     // TODO Auto-generated catch block
31     e.printStackTrace();
32 }
33 }
34 }
35
36
37
38
39
40
```

```
FileReadExampleUsingTryWithResources.java X FileWriteExample.java X FileReadExampleUsingBufferedInputStream.java X RandomAccessFileExaple.java X BufferedReaderExample.java X UserInputExample.java X
1 package io_programming;
2
3 import java.io.FileOutputStream;
4
5 public class FileWriteExample {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         String filePath = "./src/resources/cartoons.txt";
10        try(FileOutputStream fout = new FileOutputStream(filePath, true)){
11            //In the above statement 'true' indicates APPEND mode is ON
12            //By default it is OFF
13            String cartoons = "\nTom and Jerry\nDoraemon";
14            //Convert String into byte array
15            byte cartoonData[] = cartoons.getBytes();
16            fout.write(cartoonData);
17            System.out.println("Data is written to file.");
18        }
19        catch(Exception ex) {
20            ex.printStackTrace();
21        }
22    }
23
24 }
25
26 }
```

```
FileReadExampleUsingTryWithResources.java FileWriteExample.java FileReadExampleUsingBufferedInputStream.java RandomAccessFileExaple.java BufferedReaderExample.java UserInputExample.java
1 package io_programming;
2
3 import java.io.BufferedInputStream;
4 import java.io.FileInputStream;
5 import java.io.FileNotFoundException;
6 import java.io.IOException;
7 import java.io.InputStream;
8
9 public class FileReadExampleUsingBufferedInputStream {
10
11     public static void main(String[] args) {
12         // TODO Auto-generated method stub
13         String filePath = "./src/resources/technologies.txt";
14
15         try(FileInputStream fin = new FileInputStream(filePath);
16             BufferedInputStream bin = new BufferedInputStream(fin)) {
17             //Resource is initialized and instantiated
18             while(true) {
19                 //Read the content character by character using read()
20                 int data = bin.read(); //Resource is getting used
21                 //The read() method reads 1 character and returns its value and shifts the pointer 1 step ahead
22                 //When EOF is encountered, the method returns -1
23                 if(data == -1)
24                     break;
25                 char ch = (char) data;
26                 System.out.print(ch);
27             }
28         } catch (FileNotFoundException e) {
29             // TODO Auto-generated catch block
30             e.printStackTrace();
31         } catch (IOException e) {
32             // TODO Auto-generated catch block
33         }
34     }
35 }
```

FileReadExampleUsingTryWithResources.java | FileWriteExample.java | FileReadExampleUsingBufferedInputStream.java | RandomAccessFileExaple.java | BufferedReaderExample.java | UserInputExample.java

```
25         char ch = (char) data;
26         System.out.print(ch);
27     }
28 } catch (FileNotFoundException e) {
29     // TODO Auto-generated catch block
30     e.printStackTrace();
31 } catch (IOException e) {
32     // TODO Auto-generated catch block
33     e.printStackTrace();
34 }
35 }
36 }
```

```
FileReadExampleUsingTryWithResources.java FileWriteExample.java FileReadExampleUsingBufferedInputStream.java RandomAccessFileExaple.java BufferedReaderExample.java UserInputExample.java
```

```
1 package io_programming;
2
3 import java.io.File;
4 import java.io.RandomAccessFile;
5
6 public class RandomAccessFileExaple {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        String filePath = "./src/resources/cartoons.txt";
11        File currentFile = new File(filePath);
12        boolean fileAvailable =currentFile.exists();
13        if(fileAvailable) {
14            //System.out.println("Found the file");
15            try(RandomAccessFile rf = new RandomAccessFile(currentFile, "r")){
16                //In the above statement, value 'r' represents READ mode
17                //Obtain the file size
18                long size = rf.length();
19                //Obtain the midPosition
20                long midPosition = size / 2;
21                //Place the file pointer at this midPosition
22                rf.seek(midPosition);
23                while(true) {
24                    int data = rf.read();
25                    if(data == -1)
26                        break;
27                    System.out.print((char)data);
28                }
29            }
30        }
31    }
32    catch(Exception ex) {
```

```
FileReadExampleUsingTryWithResources.java FileWriteExample.java FileReadExampleUsingBufferedInputStream.java RandomAccessFileExaple.java BufferedReaderExample.java UserInputExample.java
22
23     rf.seek(midPosition);
24     while(true) {
25         int data = rf.read();
26         if(data == -1)
27             break;
28         System.out.print((char)data);
29     }
30
31 }
32 catch(Exception ex) {
33     ex.printStackTrace();
34 }
35
36 }
37 else {
38     System.out.println("The file does not exist.");
39 }
40
41 }
42
43 }
44 }
```

FileReadExampleUsingTryWithResources.java FileWriteExample.java FileReadExampleUsingBufferedInputStream.java RandomAccessFileExaple.java BufferedReaderExample.java UserInputExample.java

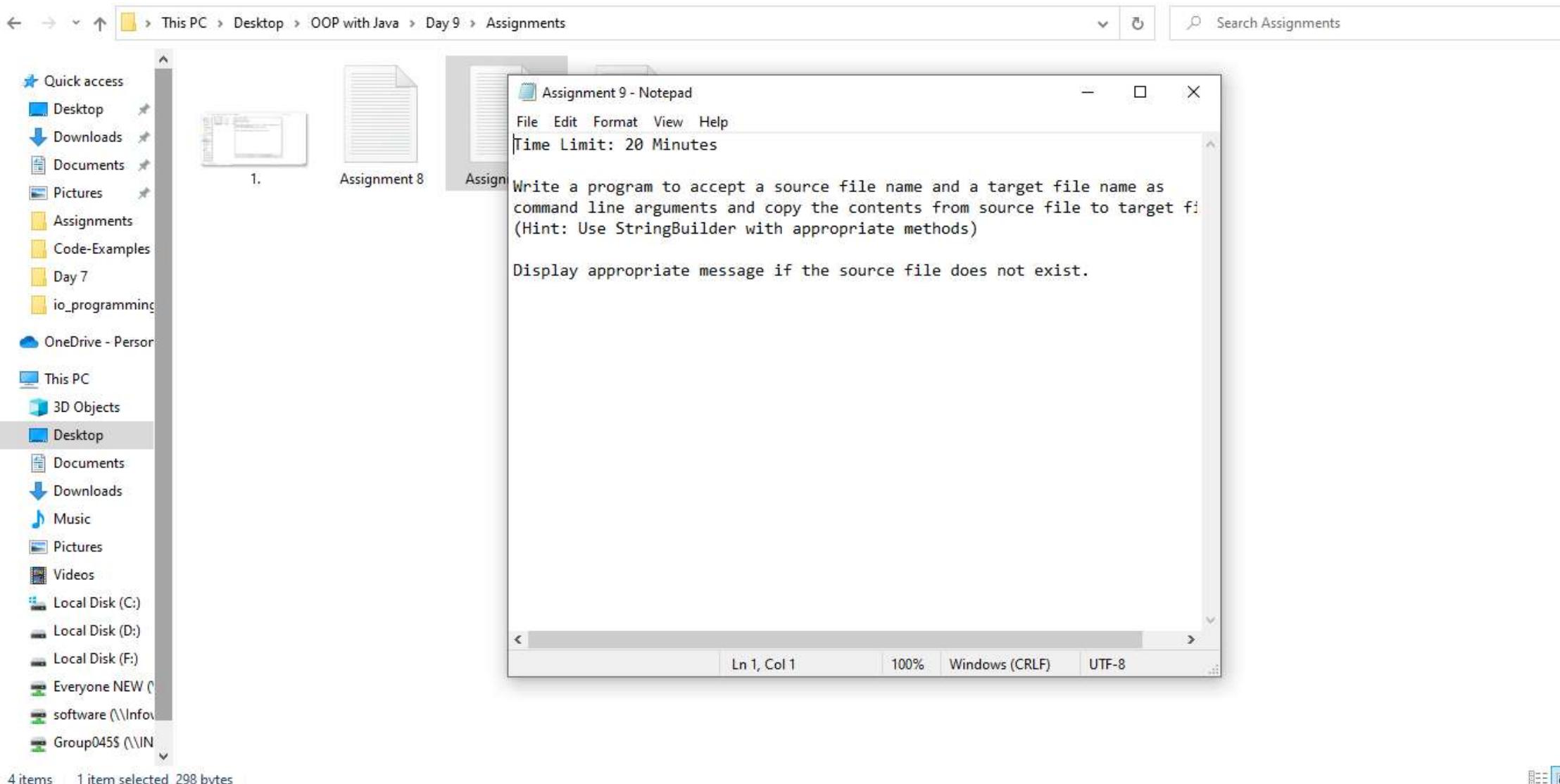
```
1 package io_programming;
2
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileReader;
6
7 public class BufferedReaderExample {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         String filePath = "./src/resources/players.txt";
12         File currentFile = new File(filePath);
13         boolean fileAvailable =currentFile.exists();
14         if(fileAvailable) {
15             try(FileReader fr = new FileReader(currentFile);
16                 BufferedReader br = new BufferedReader(fr)){
17                 while(true) {
18                     String playerInfo = br.readLine();
19                     if(playerInfo == null)//Checking for EOF
20                         break;
21                     //Split playerInfo into pieces based upon the delimiter: '='
22                     String[] playerData = playerInfo.split("=");
23                     String name = playerData[0];
24                     String nickName = playerData[1];
25                     System.out.println("Name: " + name + ", Nickname: " + nickName);
26                 }
27             }
28             catch(Exception ex) {
29                 ex.printStackTrace();
30             }
31         } else {
```

FileReadExampleUsingTryWithResources.java | FileWriteExample.java | FileReadExampleUsingBufferedInputStream.java | RandomAccessFileExaple.java | BufferedReaderExample.java | UserInputExample.java

```
25             System.out.println("Name: " + name + ", Nickname: " + nickName);
26         }
27     }
28     catch(Exception ex) {
29         ex.printStackTrace();
30     }
31 }
32 else {
33     System.out.println("The file does not exist.");
34 }
35
36 }
37
38 }
```

```
FileReadExampleUsingTryWithResources.java X FileWriteExample.java X FileReadExampleUsingBufferedInputStream.java X RandomAccessFileExaple.java X BufferedReaderExample.java X UserInputExample.java X
```

```
1 package io_programming;
2
3 import java.util.Scanner;
4
5 public class UserInputExample {
6
7     public static void main(String[] args) {
8         // Program to accept name, age and weight of a person and display the same
9         try(Scanner scr = new Scanner(System.in)){
10             System.out.println("Enter your name: ");
11             String name = scr.nextLine();
12             System.out.println("Enter age: ");
13             int age = scr.nextInt();
14             System.out.println("Enter weight: ");
15             float weight = scr.nextFloat();
16
17             System.out.println("Here are your details: ");
18             System.out.println(name + ", " + age + ", " + weight);
19         }
20         catch(Exception ex) {
21             ex.printStackTrace();
22         }
23     }
24
25 }
26
27 }
```



Time Limit: 45 Minutes

Create a file work_duration.txt that maintains a data about work done by a specific worker for the 5 days of a week in the format <>hours<>:<>minutes<>

E.g.

5:25
6:30
5:45
7:15
9:20

Write a program that fetches a data from the file and shows total duration of work done by that worker in terms of hours and minutes.

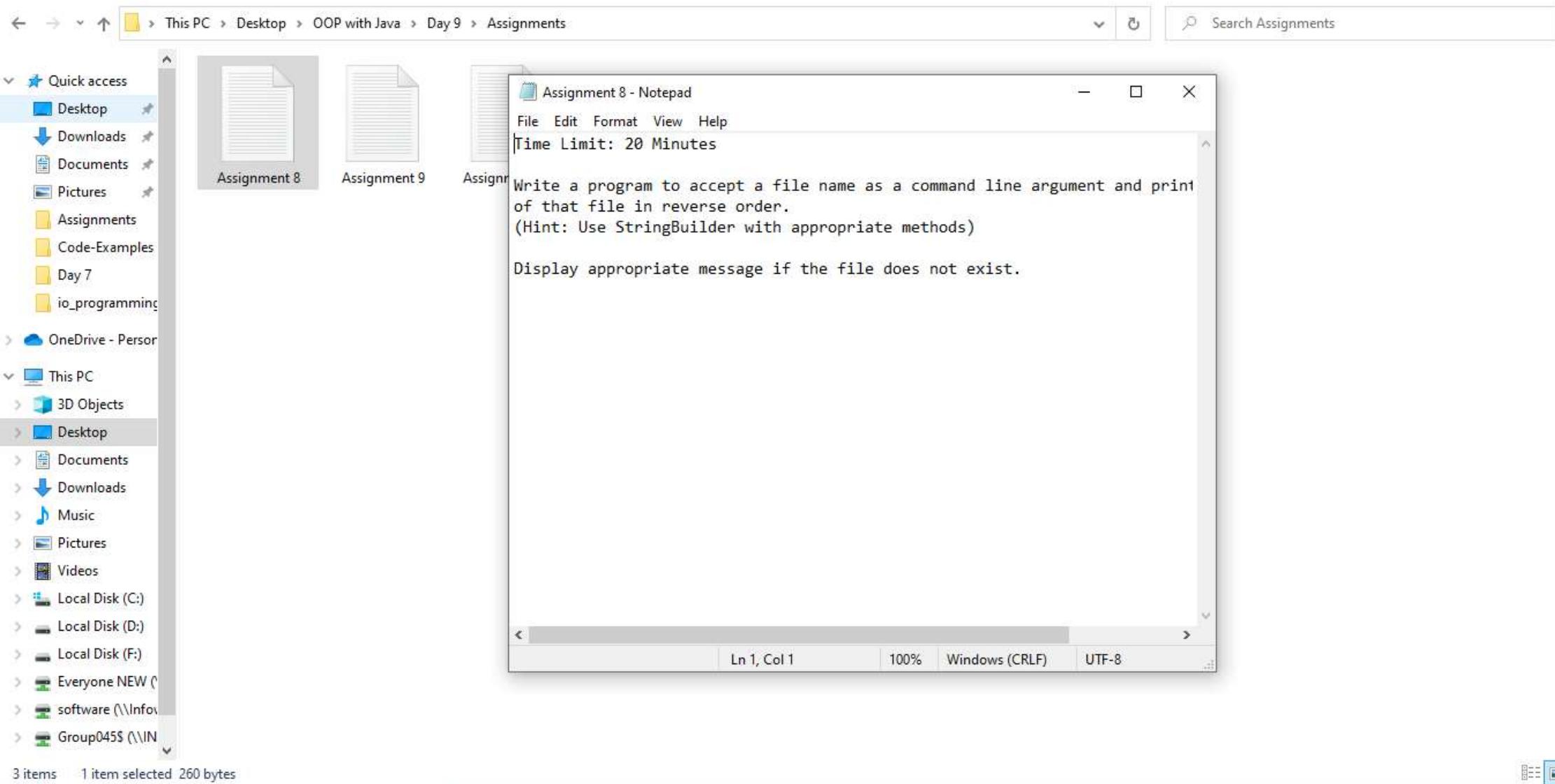
E.g.

As per the sample data, total hours = 32 and total minutes = 135 (2 hours and 15 minutes)

Hence the output should be:

Total duration: 34 hours and 15 minutes.

(Hint: Use String class appropriate method and an appropriate wrapper class)





File Edit Format View Help

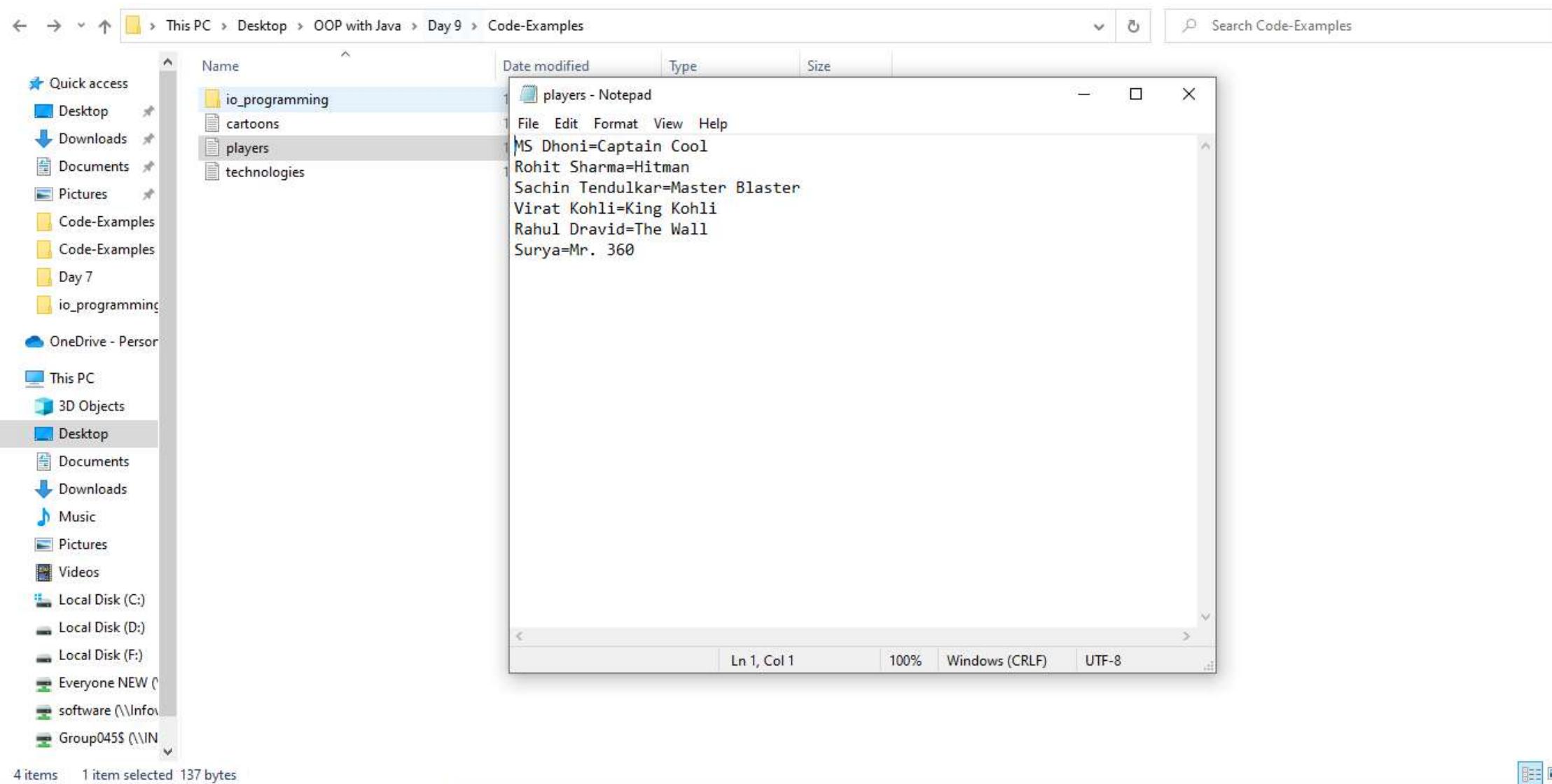
Pink Panther

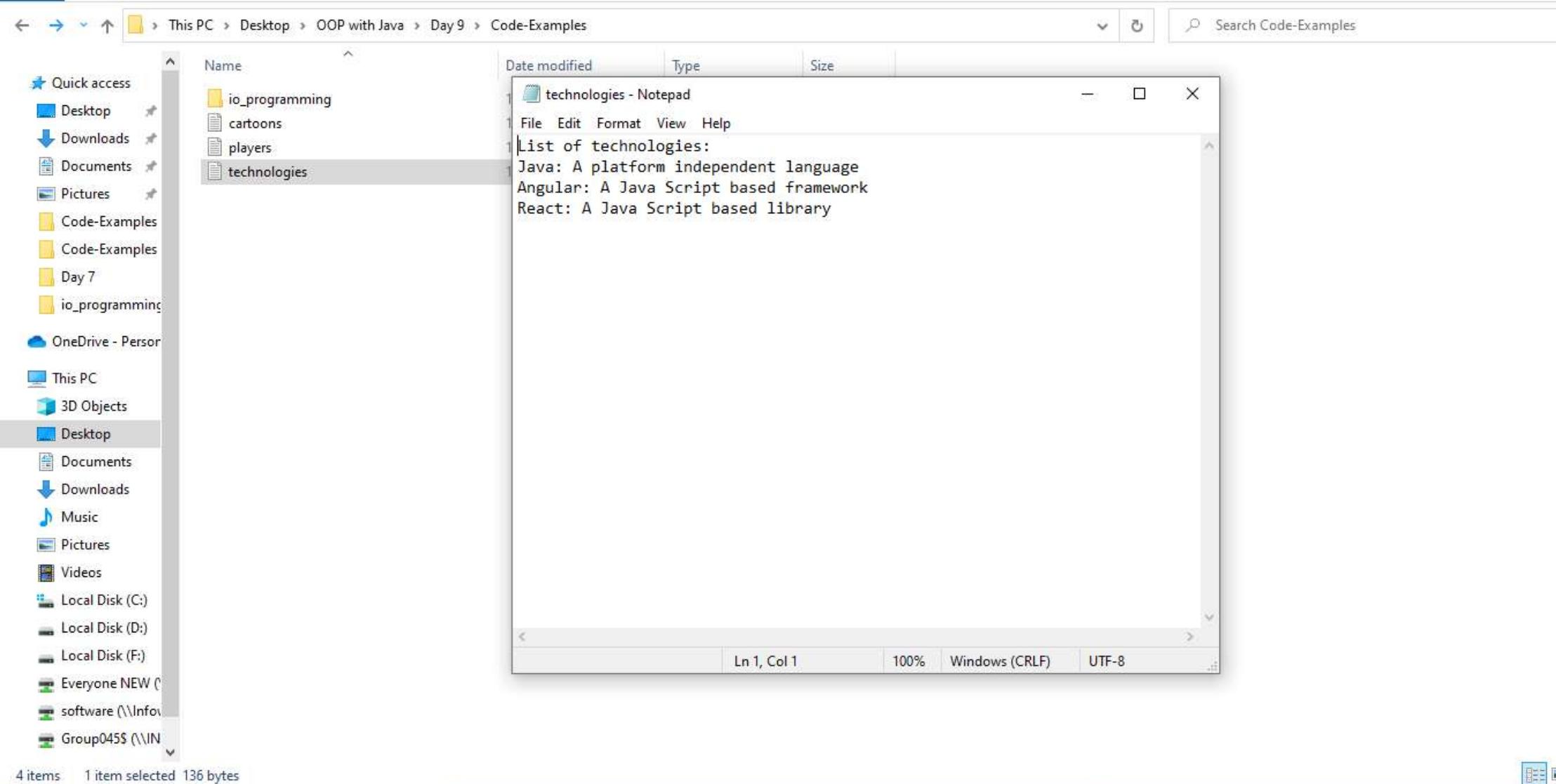
Chota Bheem

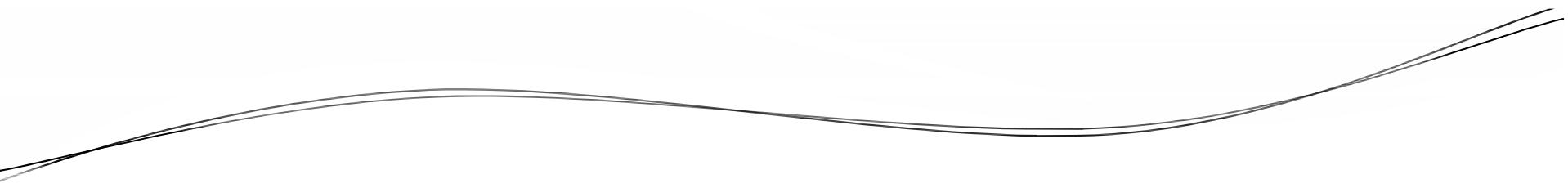
Tom and Jerry

Doraemon



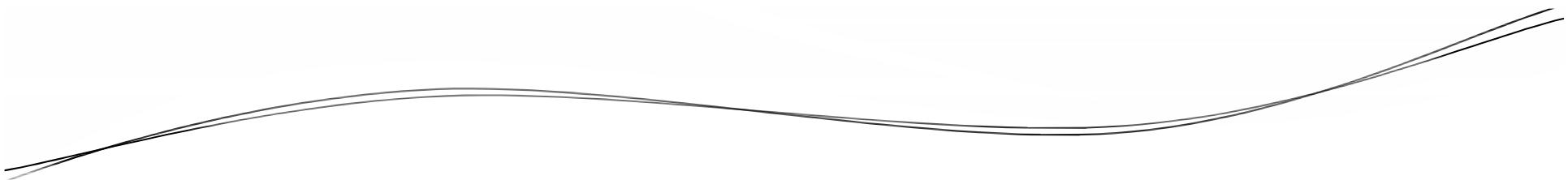






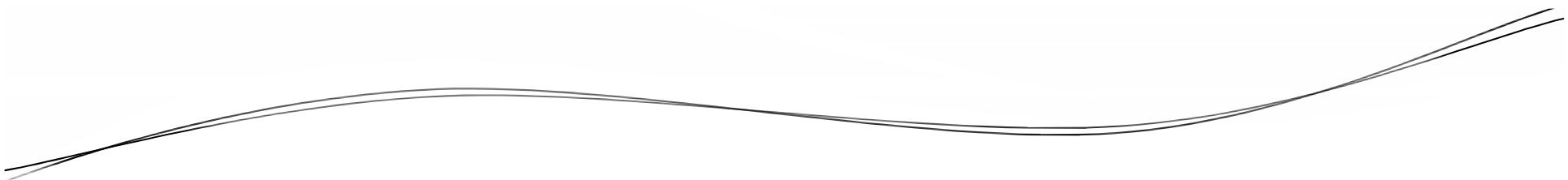
Set

By Rahul Barve



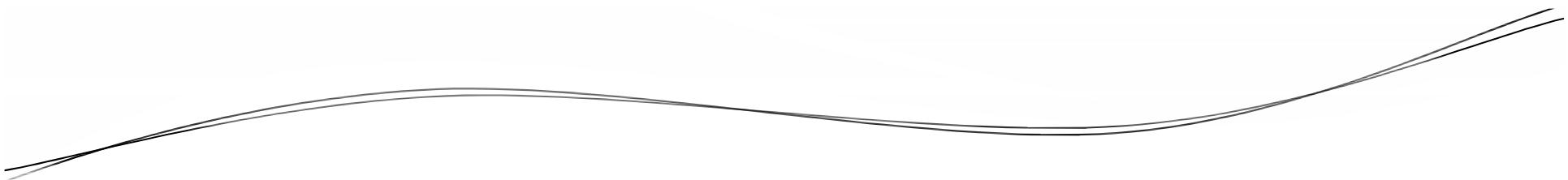
Set

- It is also inherited from Collection.
- It is an unordered collection and prevents duplicate values.



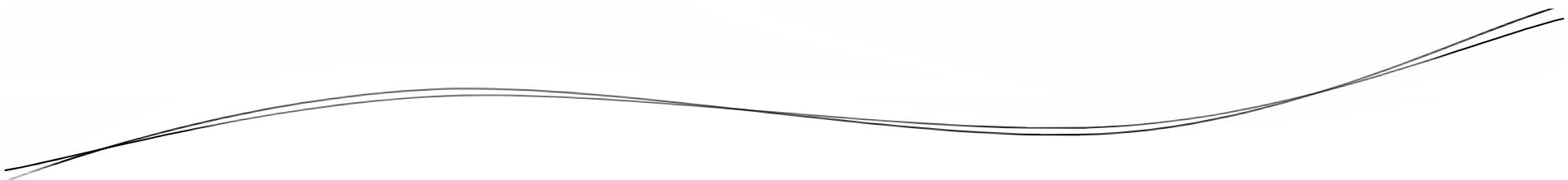
Set

- It is implemented by HashSet.
- Uses a hashing algorithm instead of index to store the elements.



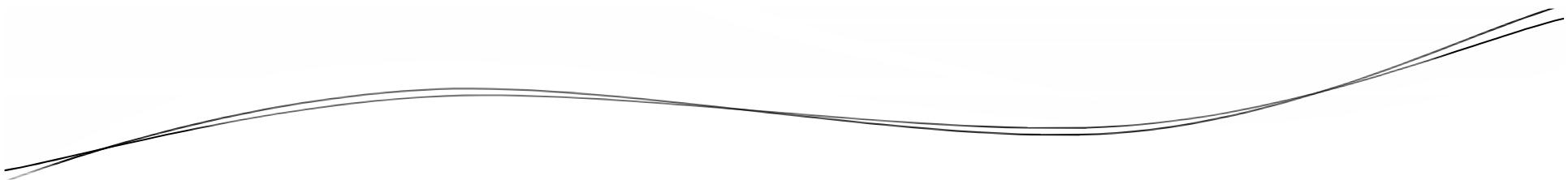
Set

- To acquire appropriate behavior of Set, the element specific class must override hashCode() and equals().



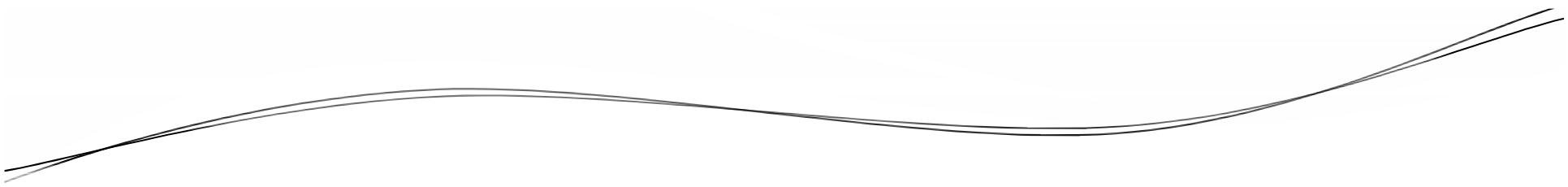
More on hashCode () and equals ()

By Rahul Barve



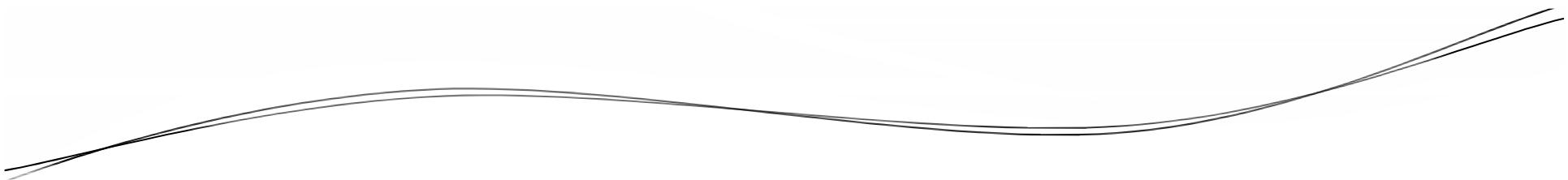
More on hashCode () and equals ()

- If two objects are equal, their hash codes are always equal whereas if two objects are unequal still they may have the same hash code.



Map

By Rahul Barve

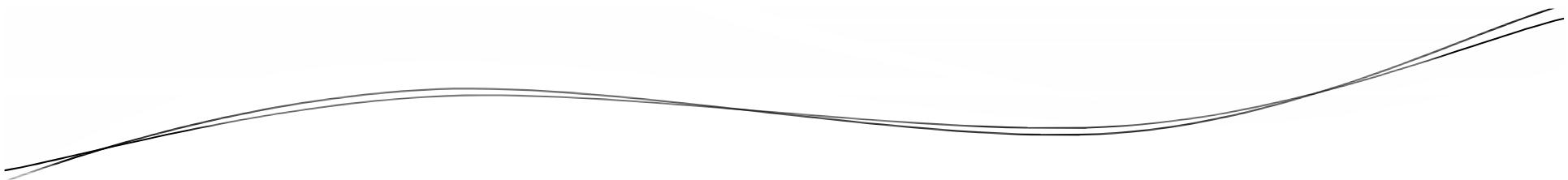


Map

- It stores elements in the form of key-value pairs.
- For every key, there is a value associated.

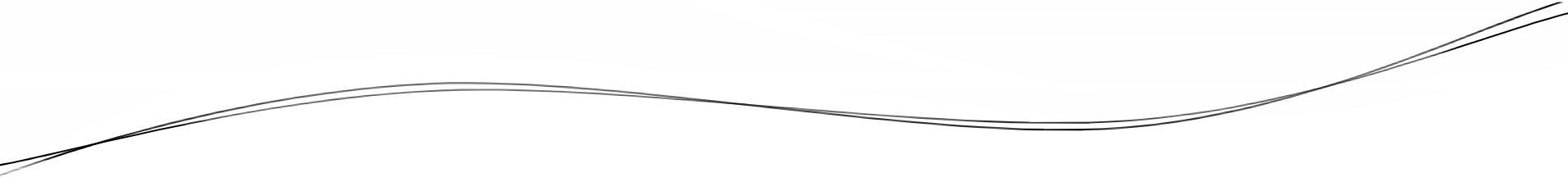
Map

- The key has to be unique, but values may be duplicates.
- Hence, the key specific class must override hashCode() and equals().



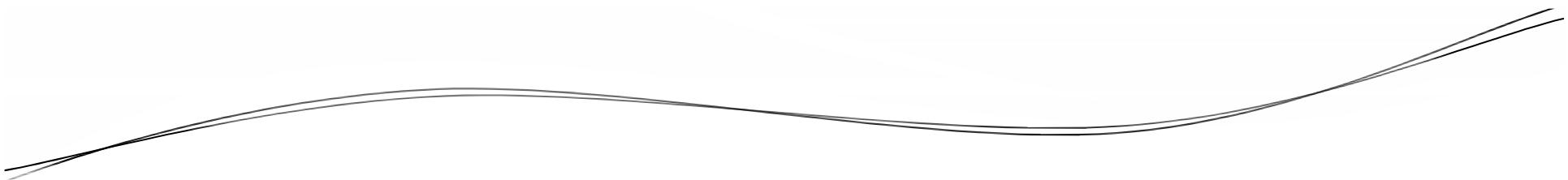
Map

- It has several Implementations:
 - Hashtable
 - HashMap
 - Properties



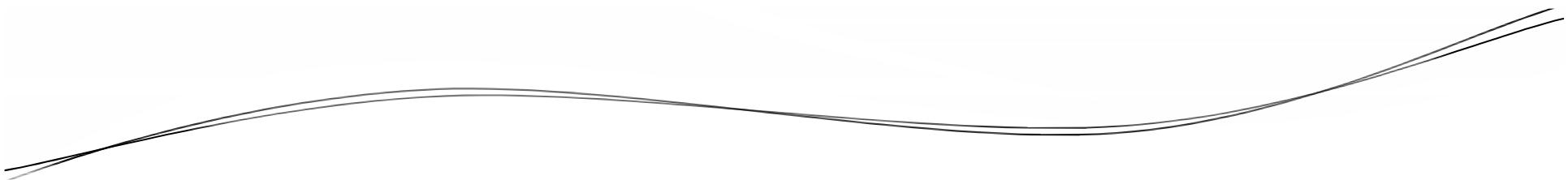
Sorted Collections

By Rahul Barve



Sorted Collections

- Whenever a collection is holding multiple data values, it is a very common requirement to perform sorting on those values.

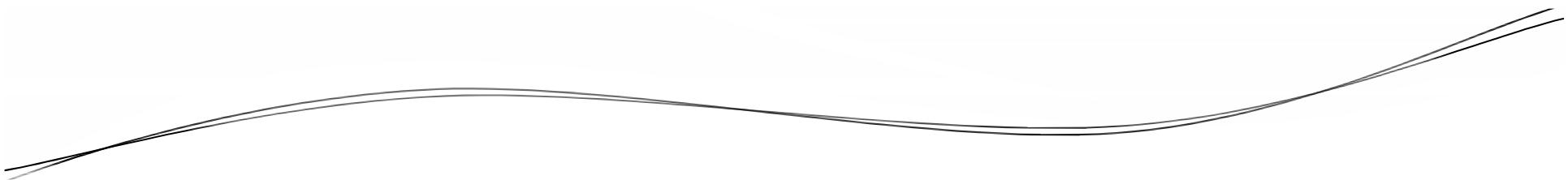


Sorted Collections

- Collections Framework provides 2 APIs to perform sorting.
- SortedSet and SortedMap

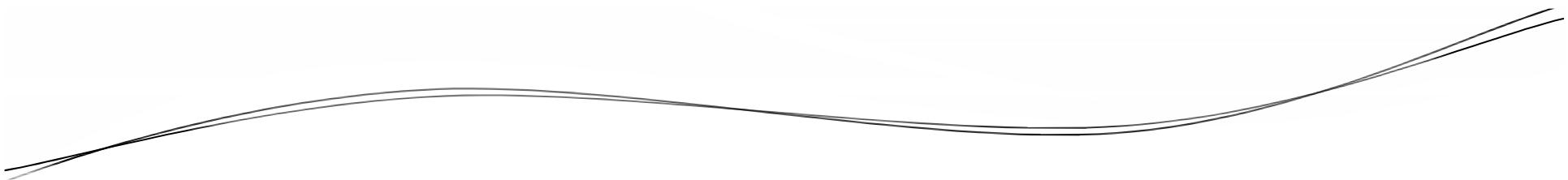
SortedSet

- It is an extension to Set.
- Does not permit duplicate values, but stores elements in a sorted order.
- It is implemented by TreeSet.



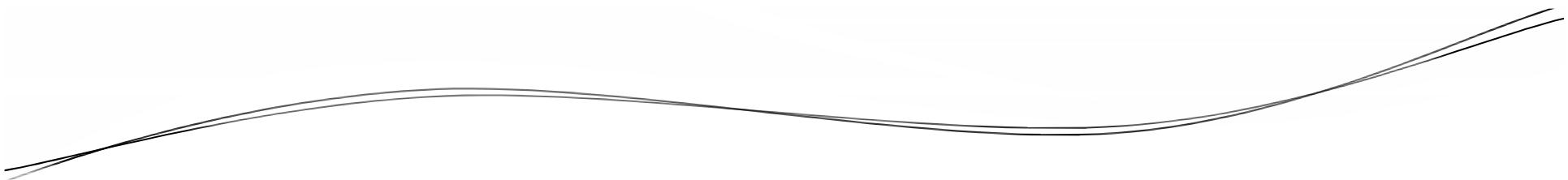
SortedMap

- It is an extension to Map.
- Stores elements in the form of key-value pairs.



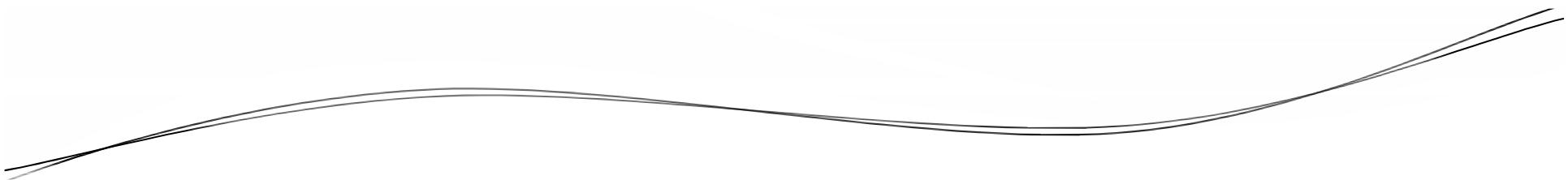
SortedMap

- Performs sorting on the basis of keys.
- It is implemented by TreeMap.



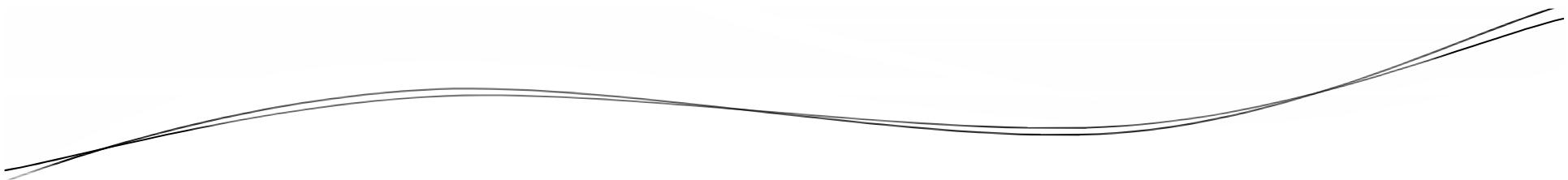
Sorting Customization

By Rahul Barve



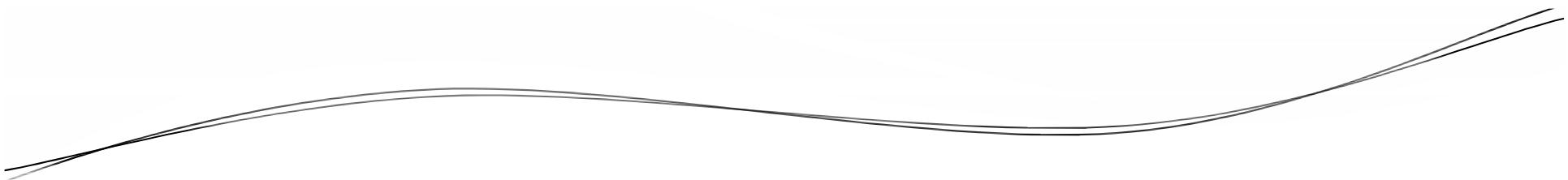
Sorting Customization

- Collection Framework also provides a mechanism to customize the sorting algorithm.
- It is especially required in case of user defined objects.



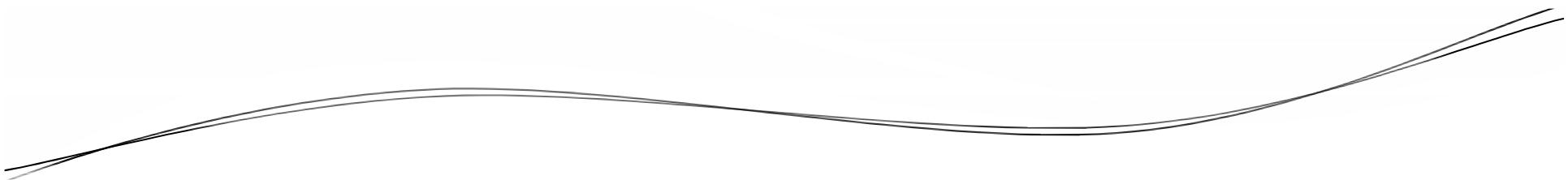
Sorting Customization

- To customize the sorting algorithm, there are 2 APIs provided:
 - `java.lang.Comparable`
 - `Java.util.Comparator`



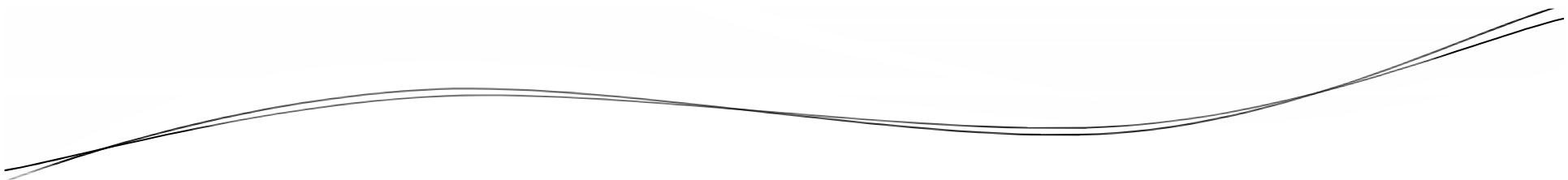
Comparable

- Implemented by a class of which objects are to be sorted.
- Useful to provide a default sorting algorithm.



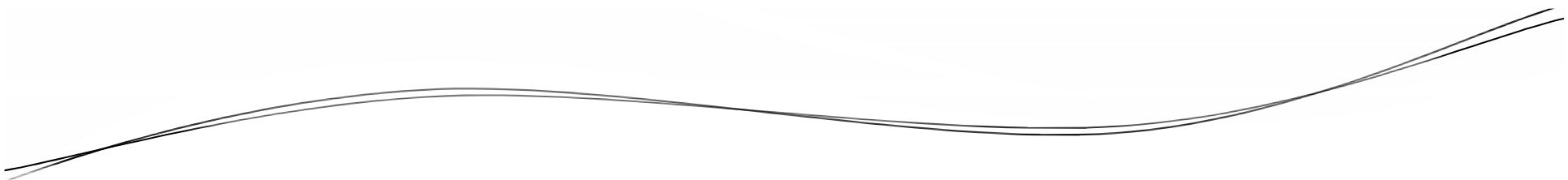
Comparator

- Implemented by a class that provides the sorting algorithm.
- Used to customize the sorting algorithm.



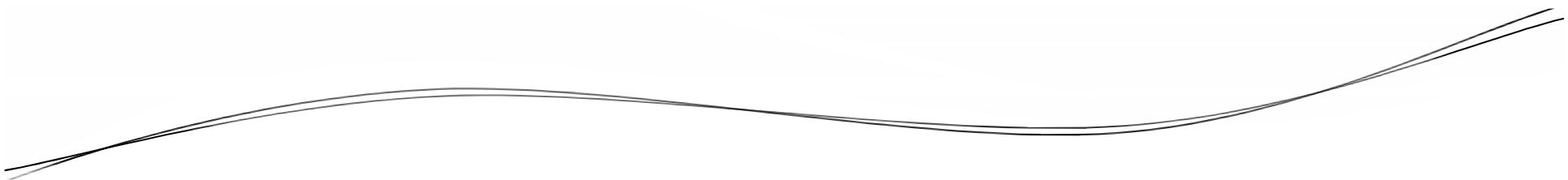
More Utility Classes

By Rahul Barve



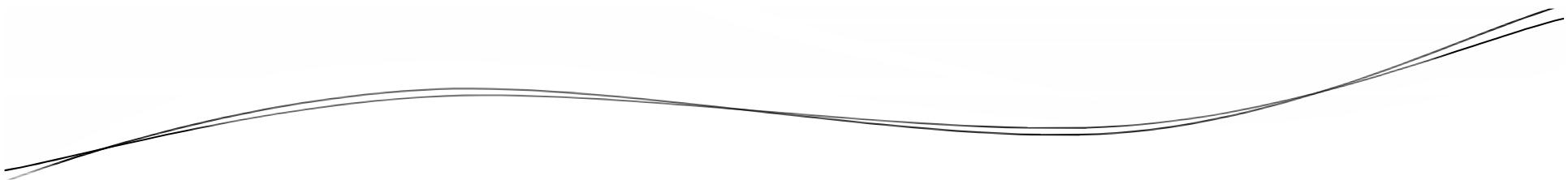
More Utility Classes

- There are several utility classes provided by `java.util` package to handle collection oriented functionalities.
- These utility classes are used to accomplish common requirements of the application



More Utility Classes

- Collections
- Arrays



Lets Summarize

- Introduction to Collections Framework.
- Understand the Need for Collections.
- Arrays Vs. Collections
- Simple Example
- Performing Iterations
- Type Safe Collections
- Types of Collections
- Working with Sorted Collections

```
StackExample.java ✘ VectorExample.java ✘ ArrayListExample.java ✘ LinkedListExample.java ✘ IterationExample.java ✘ Calculator.java ✘ StringLengthCalculator.java ✘ SquareRootCalculator.java ✘ GenericExample.java ✘ Type ✘
1 package collections;
2
3 import java.util.Stack;
4
5 public class StackExample {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         Stack values = new Stack();
10        //Adding some elements
11        values.add("Welcome");
12        values.add(true); // values.add(new Boolean(true));
13        values.push(45.23); //Double
14        values.push(100); //Integer
15        values.push(50.34f); //Float
16        int currentSize = values.size(); //Obtain the no. of elements
17        System.out.println("Current Size: " + currentSize);
18
19        Object poppedObject = values.pop(); //Removes the element from the TOP
20        int newSize = values.size();
21        System.out.println(poppedObject);
22        System.out.println("New size: " + newSize);
23
24        Object peekedObject = values.peek(); //Returns the element from the TOP but does not remove it
25        System.out.println(peekedObject);
26        int sizeAfterPeek = values.size();
27        System.out.println("Size after Peek: " + sizeAfterPeek);
28
29        System.out.println("-----Printing all the elements-----");
30        for(Object obj : values) {
31            System.out.println(obj);
32        }
33    }
34}
```

StackExample.java VectorExample.java ArrayListExample.java LinkedListExample.java IterationExample.java Calculator.java StringLengthCalculator.java SquareRootCalculator.java GenericExample.java Type

```
25     System.out.println(peekedObject);
26     int sizeAfterPeek = values.size();
27     System.out.println("Size after Peek: " + sizeAfterPeek);
28
29     System.out.println("-----Printing all the elements-----");
30     for(Object obj : values) {
31         System.out.println(obj);
32     }
33
34 }
35
36 }
37 }
```

```
StackExample.java VectorExample.java ArrayListExample.java LinkedListExample.java IterationExample.java Calculator.java StringLengthCalculator.java SquareRootCalculator.java GenericExample.java Type
1 package collections;
2
3 import java.util.Vector;
4
5 public class VectorExample {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         Vector values = new Vector();
10        int size = values.size();
11        int capacity = values.capacity();
12        System.out.println("Size: " + size + ", Capacity: " + capacity);
13        values.add("Hello");
14        values.add(200);
15        values.add(34.76);
16        size = values.size();
17        capacity = values.capacity();
18        System.out.println("Size: " + size + ", Capacity: " + capacity);
19        //Adding 8 more elements
20        for(int a=1;a<=8;a++)
21            values.add(a);
22        size = values.size();
23        capacity = values.capacity();
24        System.out.println("Size: " + size + ", Capacity: " + capacity);
25
26        System.out.println("-----Printing all the elements-----");
27        for(Object obj : values) {
28            System.out.println(obj);
29        }
30
31        System.out.println("3rd element is " + values.get(2)); //Index starts from 0 and hence 2 --> Gives 3rd element
32    }
}
```

```
StackExample.java X VectorExample.java X ArrayListExample.java X LinkedListExample.java X IterationExample.java X Calculator.java X StringLengthCalculator.java X SquareRootCalculator.java X GenericExample.java X Type X
1 package collections;
2
3 import java.util.ArrayList;
4
5 public class ArrayListExample {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         ArrayList values = new ArrayList();
10        values.add("Hello");
11        values.add(200);
12        values.add(34.76);
13
14        System.out.println("-----Printing all the elements-----");
15        for(Object obj : values) {
16            System.out.println(obj);
17        }
18
19        System.out.println("3rd element is " + values.get(2));
20
21        //Inserting an element somewhere in between
22        values.add(2, "Test");
23        System.out.println("-----");
24        for(Object obj : values) {
25            System.out.println(obj);
26        }
27    }
28
29 }
30
31
32 }
```

```
StackExample.java | VectorExample.java | ArrayListExample.java | LinkedListExample.java | IterationExample.java | Calculator.java | StringLengthCalculator.java | SquareRootCalculator.java | GenericExample.java | Type |
1 package collections;
2
3 import java.util.ArrayList;
4 import java.util.LinkedList;
5
6 public class LinkedListExample {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        LinkedList values = new LinkedList();
11        values.add("Hello");
12        values.add(200);
13        values.add(34.76);
14        values.add("Hello");
15        values.add(200);
16        values.add(34.76);
17        System.out.println("-----Printing all the elements-----");
18        for(Object obj : values) {
19            System.out.println(obj);
20        }
21
22        //Adding a new element directly at the TOP
23        values.addFirst("Good Morning");
24        System.out.println("-----Printing all the elements-----");
25        for(Object obj : values) {
26            System.out.println(obj);
27        }
28
29        //Removing an element from the TOP
30        values.removeFirst();
31        System.out.println("-----Printing all the elements-----");
32        for(Object obj : values) {
```

StackExample.java VectorExample.java ArrayListExample.java LinkedListExample.java IterationExample.java Calculator.java StringLengthCalculator.java SquareRootCalculator.java GenericExample.java Type

```
28
29         //Removing an element from the TOP
30         values.removeFirst();
31         System.out.println("-----Printing all the elements-----");
32         for(Object obj : values) {
33             System.out.println(obj);
34         }
35     }
36
37 }
```

```
StackExample.java ✘ VectorExample.java ✘ ArrayListExample.java ✘ LinkedListExample.java ✘ IterationExample.java ✘ Calculator.java ✘ StringLengthCalculator.java ✘ SquareRootCalculator.java ✘ GenericExample.java ✘ Type ✘
1 package collections;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5 import java.util.List;
6 import java.util.ListIterator;
7
8 public class IterationExample {
9
10    public static void main(String[] args) {
11        // TODO Auto-generated method stub
12        List names = new ArrayList();
13        names.add("Ram");
14        names.add("Seeta");
15        names.add("Raavan");
16        names.add("Hanuman");
17
18        //Obtaining an Iterator on the underlying collection: names
19        Iterator namesIt = names.iterator();
20        while(namesIt.hasNext()) {//Checks whether the next element is available or not
21            Object obj = namesIt.next();//Returns the next element and shifts the cursor 1 step ahead
22            System.out.println(obj);
23        }
24
25        System.out.println("-----Performing Removal of Raavan-----");
26        namesIt = names.iterator();
27        while(namesIt.hasNext()) {
28            Object obj = namesIt.next();
29            if(obj.equals("Raavan"))
30                namesIt.remove();
31        }
32        for(Object name : names)
```

```
25     System.out.println("-----Performing Removal of Raavan-----");
26     namesIt = names.iterator();
27     while(namesIt.hasNext()) {
28         Object obj = namesIt.next();
29         if(obj.equals("Raavan"))
30             namesIt.remove();
31     }
32     for(Object name : names)
33         System.out.println(name);
34
35     System.out.println("-----Performing Reverse Traversal using ListIterator-----");
36     int size = names.size();
37     ListIterator namesListIt = names.listIterator(size);
38     while(namesListIt.hasPrevious()) {
39         Object obj = namesListIt.previous();
40         System.out.println(obj);
41     }
42 }
43
44 }
```

StackExample.java | VectorExample.java | ArrayListExample.java | LinkedListExample.java | IterationExample.java | **Calculator.java** | StringLengthCalculator.java | SquareRootCalculator.java | GenericExample.java | Type |

```
1 package collections;  
2  
3 public interface Calculator<T, R> {  
4     //In this case the interface just provides a template because it is a generic interface  
5     R doCalculate(T t);  
6  
7 }  
8  
9 }
```

```
StackExample.java X VectorExample.java X ArrayListExample.java X LinkedListExample.java X IterationExample.java X Calculator.java X StringLengthCalculator.java X SquareRootCalculator.java X GenericExample.java X Type X
1 package collections;
2
3 public class StringLengthCalculator implements Calculator<String, Integer> {
4
5     @Override
6     public Integer doCalculate(String str) {
7         // TODO Auto-generated method stub
8         return str.length();
9     }
10
11 }
12
```

StackExample.java VectorExample.java ArrayListExample.java LinkedListExample.java IterationExample.java Calculator.java StringLengthCalculator.java SquareRootCalculator.java GenericExample.java Type

```
1 package collections;
2
3 public class SquareRootCalculator implements Calculator<Integer, Double> {
4
5     @Override
6     public Double doCalculate(Integer num) {
7         // TODO Auto-generated method stub
8         return Math.sqrt(num);
9     }
10
11 }
12 }
```

StackExample.java | VectorExample.java | ArrayListExample.java | LinkedListExample.java | IterationExample.java | Calculator.java | StringLengthCalculator.java | SquareRootCalculator.java | GenericExample.java | Type

```
1 package collections;
2
3 public class GenericExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Calculator<String, Integer> calc1 = new StringLengthCalculator();
8         Integer len = calc1.doCalculate("Welcome");
9
10        Calculator<Integer, Double> calc2 = new SquareRootCalculator();
11        Double squareRoot = calc2.doCalculate(50);
12        System.out.println(len);
13        System.out.println(squareRoot);
14
15    }
16
17 }
18
```

```
Calculator.java ✘ StringLengthCalculator.java ✘ SquareRootCalculator.java ✘ GenericExample.java ✘ TypeUnsafeCollectionExample.java ✘ TypeSafeCollectionExample.java ✘ Candidate.java ✘ CandidateExample.java ✘
1 package collections;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class TypeUnsafeCollectionExample {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        List countryNames = new ArrayList();
11        countryNames.add("India");
12        countryNames.add("Japan");
13        countryNames.add("Sri Lanka");
14        countryNames.add("Australia");
15        countryNames.add("South Korea");
16        countryNames.add(1000);
17        for(Object obj: countryNames) {
18            String country = (String)obj;
19            System.out.println(country.toUpperCase());
20        }
21    }
22
23 }
24
25
26
27
28
29
30
31
32
```

```
Calculator.java ✘ StringLengthCalculator.java ✘ SquareRootCalculator.java ✘ GenericExample.java ✘ TypeUnsafeCollectionExample.java ✘ TypeSafeCollectionExample.java ✘ Candidate.java ✘ CandidateExample.java ✘
1 package collections;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class TypeSafeCollectionExample {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        List<String> countryNames = new ArrayList<>();
11        //The above statement declares the collection as Type-Safe.
12        //It tells the compiler that it will contain objects of type String only.
13        countryNames.add("India");
14        countryNames.add("Japan");
15        countryNames.add("Sri Lanka");
16        countryNames.add("Australia");
17        countryNames.add("South Korea");
18        //countryNames.add(1000);
19        //Since compiler knows that the collection holds only strings, no explicit casting is required
20        for(String country : countryNames) {
21            System.out.println(country.toUpperCase());
22        }
23    }
24
25 }
26
27
28
29
30
31
32
```

```
1 package collections;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Candidate {
7     private String name;
8     private List<String> skills;
9
10    public Candidate() {
11        // TODO Auto-generated constructor stub
12        name = "";
13        skills = new ArrayList<String>();
14    }
15
16    public Candidate(String name, List<String> skills) {
17        this.name = name;
18        this.skills = skills;
19    }
20
21    public String getName() {
22        return name;
23    }
24
25    public void setName(String name) {
26        this.name = name;
27    }
28
29    public List<String> getSkills() {
30        return skills;
31    }
32}
```

Calculator.java X StringLengthCalculator.java X SquareRootCalculator.java X GenericExample.java X TypeUnsafeCollectionExample.java X TypeSafeCollectionExample.java X Candidate.java X CandidateExample.java X

```
19
20
21     }
22
23     public String getName() {
24         return name;
25     }
26
27     public void setName(String name) {
28         this.name = name;
29     }
30
31     public List<String> getSkills() {
32         return skills;
33     }
34
35     public void setSkills(List<String> skills) {
36         this.skills = skills;
37     }
38     //Providing a utility method to add one single skill at a time
39     public void addSkill(String skill) {
40         skills.add(skill); //Adding a new skill to the existing list of skills
41     }
42
43
44
45
46
47
48
49 }
```

```
Calculator.java ✘ StringLengthCalculator.java ✘ SquareRootCalculator.java ✘ GenericExample.java ✘ TypeUnsafeCollectionExample.java ✘ TypeSafeCollectionExample.java ✘ Candidate.java ✘ CandidateExample.java ✘
1 package collections;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class CandidateExample {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        Candidate cd = new Candidate();
11        cd.setName("Gautam");
12        List<String> skills = new ArrayList<String>();
13        skills.add("Java");
14        skills.add("Python");
15        skills.add("Machine Learning");
16        cd.setSkills(skills);
17
18        Candidate cd2 = new Candidate();
19        cd2.setName("Anu");
20        cd2.addSkill("Blockchain");
21        cd2.addSkill("CAD");
22
23        System.out.println("Printing the candidate names along with their skills");
24        String name1 = cd.getName();
25        List<String> skillList1 = cd.getSkills();
26
27        String name2 = cd2.getName();
28        List<String> skillList2 = cd2.getSkills();
29
30        System.out.println("Skills of " + name1 + " are: ");
31        for(String sk : skillList1)
32            System.out.println(sk);
```

```
Calculator.java ✘ StringLengthCalculator.java ✘ SquareRootCalculator.java ✘ GenericExample.java ✘ TypeUnsafeCollectionExample.java ✘ TypeSafeCollectionExample.java ✘ Candidate.java ✘ CandidateExample.java ✘
22
23     System.out.println("Printing the candidate names along with their skills");
24     String name1 = cd.getName();
25     List<String> skillList1 = cd.getSkills();
26
27     String name2 = cd2.getName();
28     List<String> skillList2 = cd2.getSkills();
29
30     System.out.println("Skills of " + name1 + " are: ");
31     for(String sk : skillList1)
32         System.out.println(sk);
33     System.out.println("-----");
34     System.out.println("Skills of " + name2 + " are: ");
35     for(String sk : skillList2)
36         System.out.println(sk);
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52     }
53
```

IO Assignment - Notepad

File Edit Format View Help

```
Create a class CricketTeam with following attributes:  
    name (String)  
    matchesPlayed (int)  
    matchesWon (int)
```

Write a main program that builds an array of CricketTeam for
3 elements, accepts an input from end user for all 3 teams
and writes the entire array to some file.

(Hint: Use Serialization)

Write another main program that loads the information from
that file and displays the names of the teams of which the
performance is Excellent.

A performance is Excellent if the WIN Percentage is 70 and
above.

(Hint: Use Deserialization)

RandomAccessFileExample.java | BufferedReaderExample.java | UserInputExample.java | NIOExample.java | SerializationExample.java | Course.java | DeserializationExample.java

```
22     return duration;
23 }
24 public void setDuration(int duration) {
25     this.duration = duration;
26 }
27 @Override
28 public String toString() {
29     return "Course [title=" + title + ", duration=" + duration + "]";
30 }
31
32
33 }
34 }
```

```
RandomAccessFileExaple.java ✘ BufferedReaderExample.java ✘ UserInputExample.java ✘ NIOExample.java ✘ SerializationExample.java ✘ Course.java ✘ DeserializationExample.java ✘
1 package io.programming;
2
3 import java.io.FileInputStream;
4 import java.io.ObjectInputStream;
5
6 public class DeserializationExample {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        String filePath = "./src/resources/course.txt";
11        try{
12            FileInputStream fin = new FileInputStream(filePath);
13            ObjectInputStream in = new ObjectInputStream(fin)
14            ) {
15                Object serializedObject = in.readObject(); //De-serializing the Course object
16                System.out.println(serializedObject);
17                System.out.println("-----");
18                //To invoke individual methods e.g. getters, type casting is required
19                Course foundCourse = (Course)serializedObject;
20                String courseTitle = foundCourse.getTitle();
21                int courseDuration = foundCourse.getDuration();
22                System.out.println("Title is " + courseTitle + " and Duration is " + courseDuration);
23            }
24            catch(Exception ex) {
25                ex.printStackTrace();
26            }
27        }
28    }
29
30 }
31
```

RandomAccessFileExample.java | BufferedReaderExample.java | UserInputExample.java | NIOExample.java | SerializationExample.java | Course.java | DeserializationExample.java

```
1 package io_programming;
2
3 import java.io.Serializable;
4
5 public class Course implements Serializable{
6     private String title;
7     private int duration;
8     public Course() {
9         // TODO Auto-generated constructor stub
10    }
11    public Course(String title, int duration) {
12        this.title = title;
13        this.duration = duration;
14    }
15    public String getTitle() {
16        return title;
17    }
18    public void setTitle(String title) {
19        this.title = title;
20    }
21    public int getDuration() {
22        return duration;
23    }
24    public void setDuration(int duration) {
25        this.duration = duration;
26    }
27    @Override
28    public String toString() {
29        return "Course [title=" + title + ", duration=" + duration + "]";
30    }
31
32}
```

Assignment 10 - Notepad
File Edit Format View Help
Time Limit: 45 Minutes

Create a file work_duration.txt that maintains a data about work done by a specific worker for the 5 days of a week in the format <>hours>>:<>minutes>>

E.g.

5:25
6:30
5:45
7:15
9:20

Write a program that fetches a data from the file and shows total duration of work done by that worker in terms of hours and minutes.

E.g.

As per the sample data, total hours = 32 and total minutes = 135 (2 hours and 15 minutes)

Hence the output should be:

Total duration: 34 hours and 15 minutes.

(Hint: Use String class appropriate method and an appropriate wrapper class)

FileReadExampleUsingTryWithResources.java X FileWriteExample.java X FileReadExampleUsingBufferedInputStream.java X RandomAccessFileExaple.java X BufferedReaderExample.java X UserInputExample.java X NIOExample.java X

```
1 package nio;
2
3 import java.io.IOException;
4 import java.nio.file.Files;
5 import java.nio.file.Path;
6 import java.nio.file.Paths;
7
8 public class NIOExample {
9
10    private static void directoryExample() throws IOException {
11        String dirPath = "./src/mywork";
12        Path currentPath = Paths.get(dirPath);
13        boolean dirAvailable = Files.exists(currentPath);
14        if(dirAvailable)
15            System.out.println("Dir exists");
16        else {
17            System.out.println("Dir does not exist. Creating it now");
18            Files.createDirectory(currentPath);
19            System.out.println("Directory created...");
20        }
21    }
22
23    private static void fileExample() throws IOException {
24        String filePath = "./src/mywork/myfile.txt";
25        Path currentPath = Paths.get(filePath);
26        boolean fileAvailable = Files.exists(currentPath);
27        if(fileAvailable)
28            System.out.println("File exists");
29        else {
30            System.out.println("File does not exist. Creating it now");
31            Files.createFile(currentPath);
32            System.out.println("File created.");
33    }
34}
```

```
FileReadExampleUsingTryWithResources.java FileWriteExample.java FileReadExampleUsingBufferedInputStream.java RandomAccessFileExaple.java BufferedReaderExample.java UserInputExample.java NIOExample.java
```

```
28         System.out.println("File exists");
29     } else {
30         System.out.println("File does not exist. Creating it now");
31         Files.createFile(currentPath);
32         System.out.println("File created.");
33     }
34 }
35 }
36 }

37 private static void writeToFileExample() throws IOException {
38     String filePath = "./src/mywork/myfile.txt";
39     Path currentPath = Paths.get(filePath);
40     String info = "Demonstrating NIO";
41     byte data[] = info.getBytes();
42     Files.write(currentPath, data);
43     System.out.println("Data is written to file");
44 }
45

46 private static void readFromFileExample() throws IOException {
47     //I want you to complete this
48 }

49

50 public static void main(String[] args) {
51     // TODO Auto-generated method stub
52     try {
53         //directoryExample();
54         //fileExample();
55         writeToFileExample();
56     } catch (IOException e) {
57         // TODO Auto-generated catch block
58         e.printStackTrace();
59     }
}
```

```
52     try {
53         //directoryExample();
54         //fileExample();
55         writeToFileExample();
56     } catch (IOException e) {
57         // TODO Auto-generated catch block
58         e.printStackTrace();
59     }
60 }
61 }
62 }
63 }
64 }
```

```
RandomAccessFileExaple.java ✘ BufferedReaderExample.java ✘ UserInputExample.java ✘ NIOExample.java ✘ SerializationExample.java ✘ Course.java ✘ DeserializationExample.java ✘
1 package io_programming;
2
3 import java.io.FileOutputStream;
4 import java.io.ObjectOutputStream;
5
6 public class SerializationExample {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        String filePath = "./src/resources/course.txt";
11        try(FileOutputStream fout = new FileOutputStream(filePath);
12            ObjectOutputStream out = new ObjectOutputStream(fout)
13            ){
14            Course cr = new Course("Spring", 5);
15            out.writeObject(cr); //Serializing the Course Object referred as 'cr'
16            System.out.println("Object serialized.");
17        }
18        catch(Exception ex) {
19            ex.printStackTrace();
20        }
21    }
22
23 }
24
25 }
```

```
1 package collections;
2
3 import java.util.HashSet;
4 import java.util.Set;
5
6 public class HashSetExample {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        Set<String> fruitNames = new HashSet<String>();
11        fruitNames.add("Mango");
12        fruitNames.add("Apple");
13        fruitNames.add("Banana");
14        fruitNames.add("Watermelon");
15        fruitNames.add("Orange");
16        fruitNames.add("Banana");
17        fruitNames.add("Watermelon");
18        fruitNames.add("Banana");
19        fruitNames.add("Watermelon");
20
21        System.out.println("Printing fruit names: ");
22        for(String fruit : fruitNames)
23            System.out.println(fruit);
24    }
25
26
27
28
29
30
31
32
```

```
1 package collections;
2
3 import java.util.HashSet;
4 import java.util.Set;
5
6 public class HashSetUsingUserDefinedObjectExample {
7
8     public static void main(String[] args) {
9         Set<Fruit> allFruits = new HashSet<Fruit>();
10        //Creating 3 individual Fruit objects
11        Fruit mango = new Fruit();
12        Fruit apple = new Fruit("Apple", "Healthy");
13        Fruit watermelon = new Fruit("Watermelon", "Perfect for Summer");
14
15        //Adding them into HashSet: allFruits
16        allFruits.add(watermelon);
17        allFruits.add(mango);
18        allFruits.add(apple);
19
20        System.out.println("Printing all fruits: ");
21        for(Fruit currentFruit : allFruits)
22            System.out.println(currentFruit);
23
24        Fruit mangoAgain = new Fruit();
25        Fruit appleAgain = new Fruit("Apple", "Healthy");
26        //Adding the above objects into HashSet: allFruits
27        allFruits.add(appleAgain);
28        allFruits.add(mangoAgain);
29        System.out.println("Printing all the fruits again: ");
30        for(Fruit currentFruit : allFruits)
31            System.out.println(currentFruit);
32    }
}
```

HashSetExample.java HashSetUsingUserDefinedObjectExample.java Fruit.java HashMapExample.java PropertiesExample.java window.properties TreeSetExample.java Movie.java MovieComparatorYearWiseAscend

```
22         System.out.println(currentFruit);

23         Fruit mangoAgain = new Fruit();
24         Fruit appleAgain = new Fruit("Apple", "Healthy");
25         //Adding the above objects into HashSet: allFruits
26         allFruits.add(appleAgain);
27         allFruits.add(mangoAgain);
28         System.out.println("Printing all the fruits again: ");
29         for(Fruit currentFruit : allFruits)
30             System.out.println(currentFruit);

31     }

32 }

33 }

34 }

35 }

36 }

37 }

38 }

39 }

40 }

41 }

42 }

43 }
```

```
1 package collections;
2
3 import java.util.Objects;
4
5 public class Fruit {
6     private String name;
7     private String description;
8     public Fruit() {
9         name = "Mango";
10        description = "King";
11    }
12    public Fruit(String name, String description) {
13        this.name = name;
14        this.description = description;
15    }
16    public String getName() {
17        return name;
18    }
19    public void setName(String name) {
20        this.name = name;
21    }
22    public String getDescription() {
23        return description;
24    }
25    public void setDescription(String description) {
26        this.description = description;
27    }
28    @Override
29    public String toString() {
30        return "Fruit [name=" + name + ", description=" + description + "]";
31    }
32}
```

HashSetExample.java HashSetUsingUserDefinedObjectExample.java Fruit.java HashMapExample.java PropertiesExample.java window.properties TreeSetExample.java Movie.java MovieComparatorYearWiseAscend

```
28     @Override
29     public String toString() {
30         return "Fruit [name=" + name + ", description=" + description + "]";
31     }
32     @Override
33     public int hashCode() {
34         System.out.println("hashCode() invoked...");
35         return Objects.hash(description, name);
36     }
37     @Override
38     public boolean equals(Object obj) {
39         System.out.println("equals) invoked()...");
40         if (this == obj)
41             return true;
42         if (obj == null)
43             return false;
44         if (getClass() != obj.getClass())
45             return false;
46         Fruit other = (Fruit) obj;
47         return Objects.equals(description, other.description) && Objects.equals(name, other.name);
48     }
49
50
51
52 }
```

```
1 package collections;
2
3 import java.util.Collection;
4 import java.util.HashMap;
5 import java.util.Map;
6 import java.util.Set;
7
8 public class HashMapExample {
9
10    public static void main(String[] args) {
11        Map<String, Fruit> fruitsData = new HashMap<String, Fruit>();
12        Fruit mango = new Fruit();
13        Fruit apple = new Fruit("Apple", "Healthy");
14        Fruit watermelon = new Fruit("Watermelon", "Perfect for Summer");
15
16        //Putting all the 3 fruits as values with the associated keys as fruit codes
17        fruitsData.put("fruit1", mango);
18        fruitsData.put("fruit2", apple);
19        fruitsData.put("fruit3", watermelon);
20
21        //Fetching the information according to the keys
22        Set<String> fruitCodes = fruitsData.keySet(); //Returns Set of keys
23        for(String code : fruitCodes) {
24            //Pass the fruit code to get the fruit value
25            Fruit fruit = fruitsData.get(code);
26            System.out.println("Key: " + code + ", Value: " + fruit);
27        }
28        System.out.println("-----");
29        //Fetching values directly
30        Collection<Fruit> fruitValues = fruitsData.values(); //Returns a Collection of values
31        for(Fruit fr : fruitValues)
32            System.out.println(fr);
```

```
16 //Putting all the 3 fruits as values with the associated keys as fruit codes
17 fruitsData.put("fruit1", mango);
18 fruitsData.put("fruit2", apple);
19 fruitsData.put("fruit3", watermelon);
20
21 //Fetching the information according to the keys
22 Set<String> fruitCodes = fruitsData.keySet(); //Returns Set of keys
23 for(String code : fruitCodes) {
24     //Pass the fruit code to get the fruit value
25     Fruit fruit = fruitsData.get(code);
26     System.out.println("Key: " + code + ", Value: " + fruit);
27 }
28 System.out.println("-----");
29 //Fetching values directly
30 Collection<Fruit> fruitValues = fruitsData.values(); //Returns a Collection of values
31 for(Fruit fr : fruitValues)
32     System.out.println(fr);
33
34 }
35
36 }
37
38
39
40
41
42
43
44
```

```
1 package collections;
2
3 import java.io.FileInputStream;
4 import java.util.Enumeration;
5 import java.util.Properties;
6
7 public class PropertiesExample {
8
9     public static void main(String[] args) {
10         Properties winProps = new Properties();
11         //Load the data from window.properties into the Properties object: winProps
12         String filePath = "./src/resources/window.properties";
13         try(FileInputStream fin = new FileInputStream(filePath)){
14             winProps.load(fin);
15             //After the above statement's execution, all the information from the file is loaded into
16             //Properties object: winProps
17             Enumeration propertyNames = winProps.propertyNames();
18             while(propertyNames.hasMoreElements()) {
19                 Object obj = propertyNames.nextElement();
20                 String propertyName = (String)obj;
21                 String propertyValue = winProps.getProperty(propertyName);
22                 System.out.println(propertyName + " -----> " + propertyValue);
23             }
24         }
25         catch(Exception ex) {
26             ex.printStackTrace();
27         }
28     }
29 }
30
31 }
32 }
```

HashSetExample.java HashSetUsingUserDefinedObjectExample.java Fruit.java HashMapExample.java PropertiesExample.java window.properties TreeSetExample.java Movie.java MovieComparatorYearWiseAscend

```
1 title=My New Window
2 width=600
3 height=400
```

HashSetExample.java HashSetUsingUserDefinedObjectExample.java Fruit.java HashMapExample.java PropertiesExample.java window.properties TreeSetExample.java Movie.java MovieComparatorYearWiseAscend

```
1 package collections;
2
3 import java.util.SortedSet;
4 import java.util.TreeSet;
5
6 public class TreeSetExample {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        SortedSet<String> movieNames = new TreeSet<>();
11        movieNames.add("Speed");
12        movieNames.add("Predator");
13        movieNames.add("Drishyam 2");
14        movieNames.add("Unchai");
15        movieNames.add("Lagaan");
16        //movieNames.add(100);
17        System.out.println("Printing all the movie names: ");
18        for(String name : movieNames)
19            System.out.println(name);
20
21    }
22
23 }
24 }
```

```
1 package collections;
2
3 public class Movie implements Comparable<Movie>{
4     private String title;
5     private int year;
6     public Movie() {
7         title = "Drishyam 2";
8         year = 2022;
9     }
10    public Movie(String title, int year) {
11        this.title = title;
12        this.year = year;
13    }
14    public String getTitle() {
15        return title;
16    }
17    public void setTitle(String title) {
18        this.title = title;
19    }
20    public int getYear() {
21        return year;
22    }
23    public void setYear(int year) {
24        this.year = year;
25    }
26    @Override
27    public String toString() {
28        return "Movie [title=" + title + ", year=" + year + "]";
29    }
30    @Override
31    public int compareTo(Movie secondMovie) {
32        // This method is used to provide the default sorting algorithm
```

```
25     }
26     @Override
27     public String toString() {
28         return "Movie [title=" + title + ", year=" + year + "]";
29     }
30     @Override
31     public int compareTo(Movie secondMovie) {
32         // This method is used to provide the default sorting algorithm
33         //E.g. Right now it is title wise Descending
34         String title1 = title;//this.title;
35         String title2 = secondMovie.title;
36         int comparison = title2.compareTo(title1);//Results into Descending Order
37         //int comparison = title1.compareTo(title2);//Results into Ascending Order
38         if(comparison == 0)
39             comparison = 1;
40         return comparison;
41     }
42
43 }
44
45
46
47
48
49
50
51 }
```

HashSetUsingUserDefinedObjectExample.java | Fruit.java | HashMapExample.java | PropertiesExample.java | window.properties | TreeSetExample.java | Movie.java | MovieComparatorYearWiseAscending.java | MovieCompar...

```
1 package collections;
2
3 import java.util.Comparator;
4
5 public class MovieComparatorYearWiseAscending implements Comparator<Movie>{
6
7     @Override
8     public int compare(Movie movie1, Movie movie2) {
9         // This class provides a customized sorting algorithm
10        //E.g. In this case it is year wise Ascending
11        Integer year1 = movie1.getYear();
12        Integer year2 = movie2.getYear();
13        int comparison = year1.compareTo(year2);
14        return comparison;
15    }
16
17 }
18
19
20
21 }
```

Fruit.java HashMapExample.java PropertiesExample.java window.properties TreeSetExample.java Movie.java MovieComparatorYearWiseAscending.java MovieComparatorYearWiseDescending.java TreeSetUsingUs

```
1 package collections;
2
3 import java.util.Comparator;
4
5 public class MovieComparatorYearWiseDescending implements Comparator<Movie>{
6
7     @Override
8     public int compare(Movie movie1, Movie movie2) {
9         // This class provides a customized sorting algorithm
10        //E.g. In this case it is year wise Ascending
11        Integer year1 = movie1.getYear();
12        Integer year2 = movie2.getYear();
13        int comparison = year2.compareTo(year1);
14        return comparison;
15    }
16}
17
18
19
20
21
```

```
PropertiesExample.java window.properties TreeSetExample.java Movie.java MovieComparatorYearWiseAscending.java MovieComparatorYearWiseDescending.java TreeSetUsingUserDefinedObjectExample.java UtilityClasse
1 package collections;
2
3 import java.util.Comparator;
4 import java.util.SortedSet;
5 import java.util.TreeSet;
6
7 public class TreeSetUsingUserDefinedObjectExample {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         Comparator<Movie> comp1 = new MovieComparatorYearWiseAscending();
12         Comparator<Movie> comp2 = new MovieComparatorYearWiseDescending();
13         //SortedSet<Movie> allMovies = new TreeSet<Movie>(); //Uses Default Sorting Algorithm
14         SortedSet<Movie> allMovies = new TreeSet<Movie>(comp2);
15         //In the above statement, customized sorting algorithm is used which is given by either comp1 or comp2
16         Movie m1 = new Movie();
17         Movie m2 = new Movie("Speed", 1997);
18         Movie m3 = new Movie("Predator", 1990);
19         Movie m4 = new Movie("Lagaan", 2000);
20         Movie m5 = new Movie("PK", 2014);
21         allMovies.add(m1);
22         allMovies.add(m2);
23         allMovies.add(m3);
24         allMovies.add(m4);
25         allMovies.add(m5);
26
27         System.out.println("Printing all movies: ");
28         for(Movie movie : allMovies)
29             System.out.println(movie);
30
31     }
32 }
```

```
PropertiesExample.java x window.properties x TreeSetExample.java x Movie.java x MovieComparatorYearWiseAscending.java x MovieComparatorYearWiseDescending.java x TreeSetUsingUserDefinedObjectExample.java x UtilityClasse x
13 //SortedSet<Movie> allMovies = new TreeSet<Movie>(); //Uses Default Sorting Algorithm
14 SortedSet<Movie> allMovies = new TreeSet<Movie>(comp2);
15 //In the above statement, customized sorting algorithm is used which is given by either comp1 or comp2
16 Movie m1 = new Movie();
17 Movie m2 = new Movie("Speed", 1997);
18 Movie m3 = new Movie("Predator", 1990);
19 Movie m4 = new Movie("Lagaan", 2000);
20 Movie m5 = new Movie("PK", 2014);
21 allMovies.add(m1);
22 allMovies.add(m2);
23 allMovies.add(m3);
24 allMovies.add(m4);
25 allMovies.add(m5);

26 System.out.println("Printing all movies: ");
27 for(Movie movie : allMovies)
28     System.out.println(movie);
29
30 }
31
32 }
33
34
35
36
37
38
39
40
41
42
```

```
1 package collections;
2
3 import java.util.Arrays;
4 import java.util.Collections;
5 import java.util.List;
6
7 public class UtilityClassesExample {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         List<String> socialMedia = Arrays.asList("Twitter", "Instagram", "Whatsapp", "Facebook");
12         for(String sm : socialMedia)
13             System.out.println(sm);
14
15         System.out.println("----Sorting the elements-----");
16         Collections.sort(socialMedia);
17         for(String sm : socialMedia)
18             System.out.println(sm);
19
20     }
21
22 }
23
24
25
26
27
28
29
30
```

GenericExample.java TypeUnsafeCollectionExample.java TypeSafeCollectionExample.java Candidate.java CandidateExample.java Assignment 11.txt Assignment 12.txt Assignment 13.txt

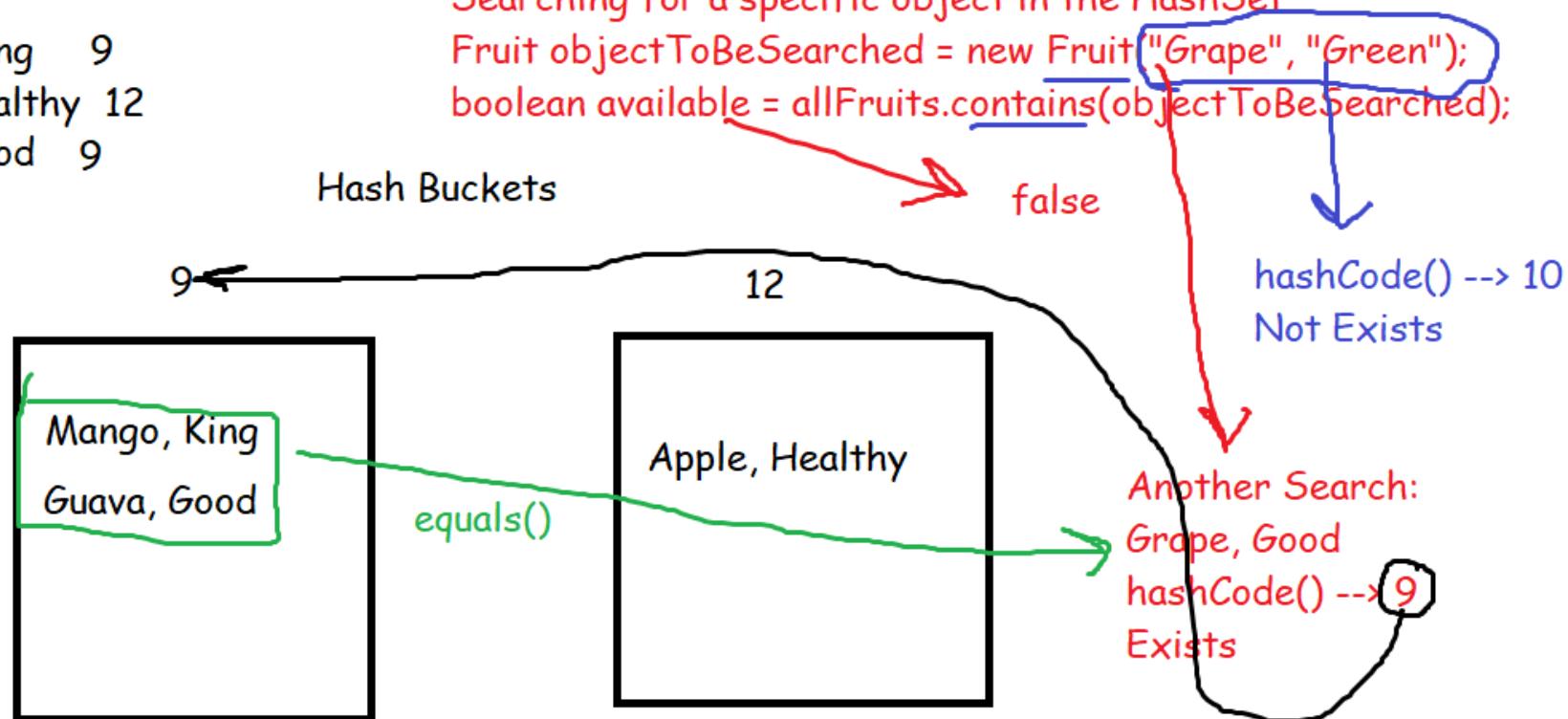
```
1 Time Limit: 45 Minutes
2
3 Create a class Score with following attributes:
4     int runs
5     boolean notOut
6 Create another class Batter with following attributes:
7     String name
8     List<Score> scores
9
10 Implement an additional method in the Batter class as per the following:
11     public float getBattingAverage()
12
13 Write a main program that creates at least 2 objects of Batter and prints their names along
14 with batting averages.
15
16 (Note: If the Batter remains NOTOUT through out the series, batting average must be displayed
17     as "Infinity". To handle this, write any algorithm of your choice.
18     Scores are for 5 match series.)
19
20 Total Runs: 165
21 Not Out: 2 Times then Avg: 55
22 Not Out: 0 times then Avg: 33
23 Not Out: 5 times then Avg: Infinity
24
```

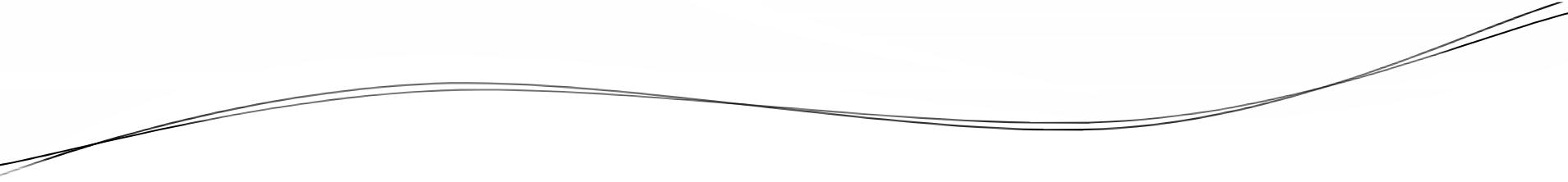
```
1 Time Limit: 45 Minutes
2 Create a class User with following attributes:
3     userName (String)
4     password (String)
5 Create a class UserDetails with following attributes:
6     firstName (String)
7     lastName (String)
8     age (int)
9
10 Create a class UserCollection with following attribute:
11     static Map<User, UserDetails> userData;
12
13 Implement a 'static' block in the class to initialize the map: 'userData'
14 with some key-value pairs.
15
16 Implement a method as per the following:
17     UserDetails getUserDetails(User key)
18
19 Write a main program that accepts a User Input as credentials
20     (Username and Password)
21
22 The program must perform following operations:
23     1. Display all the details of the user if the user is valid
24     2. If invalid, fire an exception InvalidUserException and
25         display appropriate message.
26
27 (Note: The exception must be a checked exception
28
```

GenericExample.java TypeUnsafeCollectionExample.java TypeSafeCollectionExample.java Candidate.java CandidateExample.java Assignment 11.txt Assignment 12.txt Assignment 13.txt

```
1 Time Limit: 45 Minutes
2
3 Modify the previous assignment in order to print the all
4 the details of all the users in ascending order as per their ages.
5
6 If age is same, then it must be on the basis of firstName
7
8
```

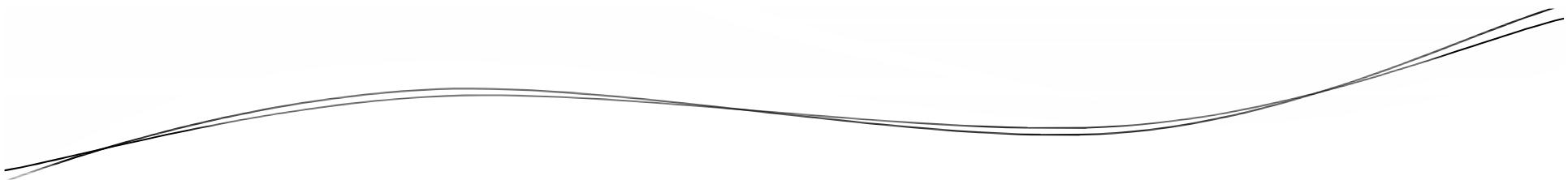
Mango, King 9
Apple, Healthy 12
Guava, Good 9





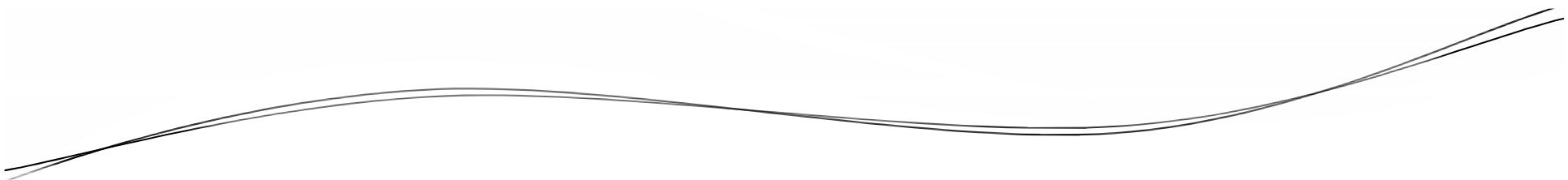
Language Essentials

By Rahul Barve



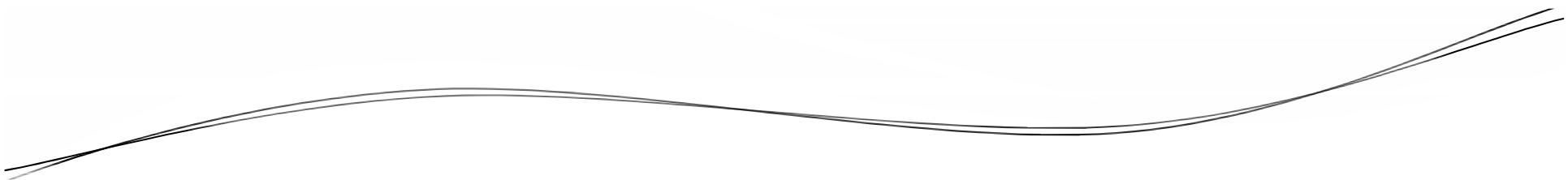
Objectives

- Inner Classes
- Using Enums
- Reflection API
- Lambda Expressions
- Date and Time API



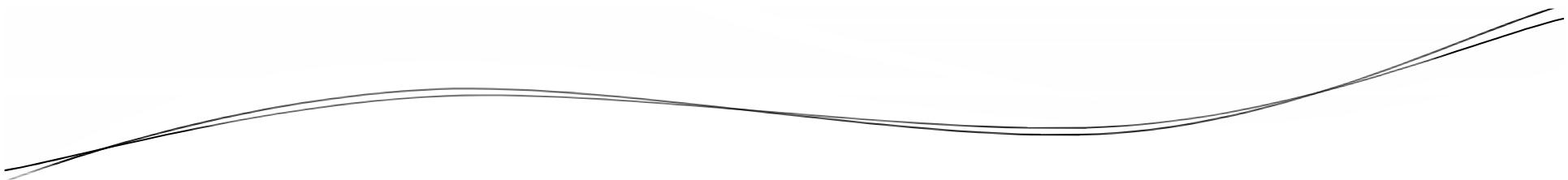
Introduction to Inner Classes

By Rahul Barve



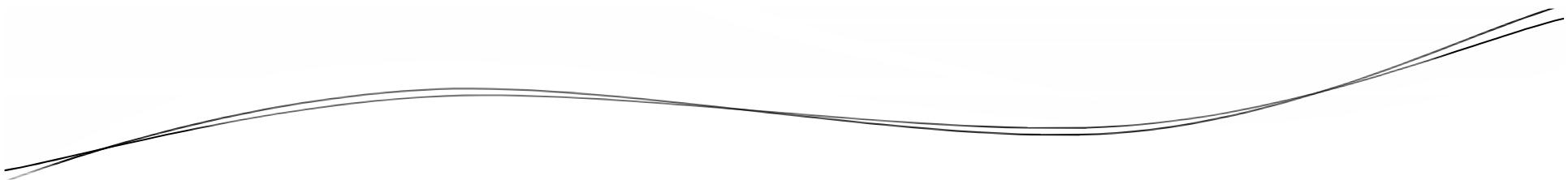
Introduction to Inner Classes

- In Java, it is possible to declare a class inside another class.
- Such a class is known as an inner class.



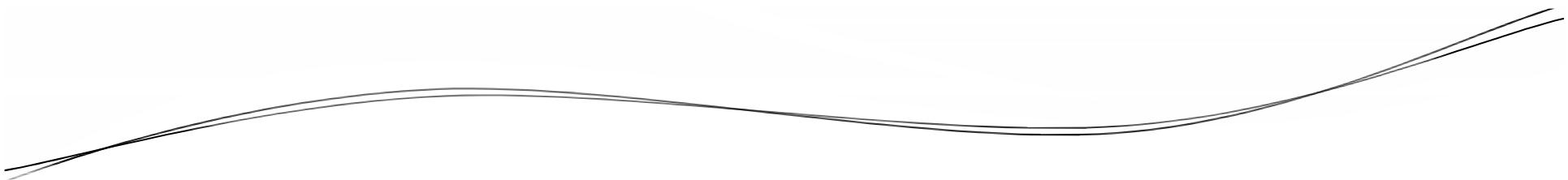
Introduction to Inner Classes

```
public class OuterClass {  
    ... . . .  
    public class InnerClass {  
        ... . . .  
    }  
}
```



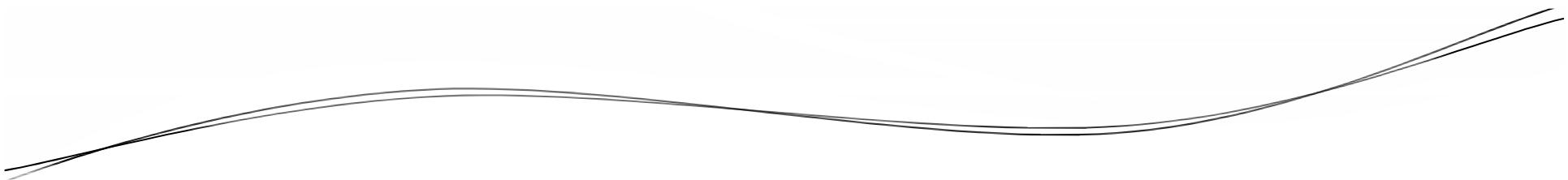
Why Inner Classes

By Rahul Barve



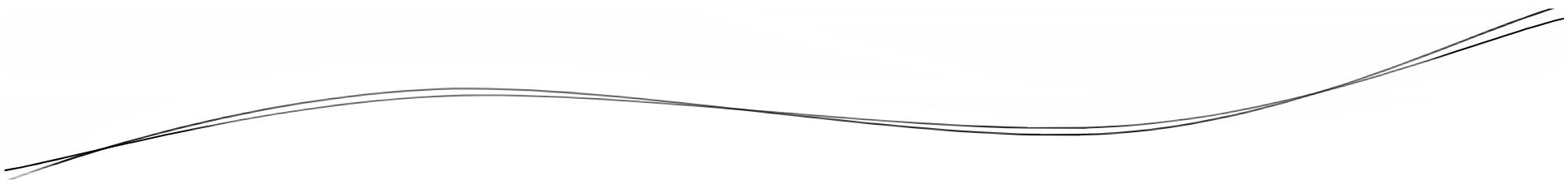
Why Inner Classes

- Inner classes are useful when a complex logic is to be isolated but handled locally within the enclosing class.
- This enables enclosing classes to handle the operations without the overhead of method invocations, sending parameters and so on.



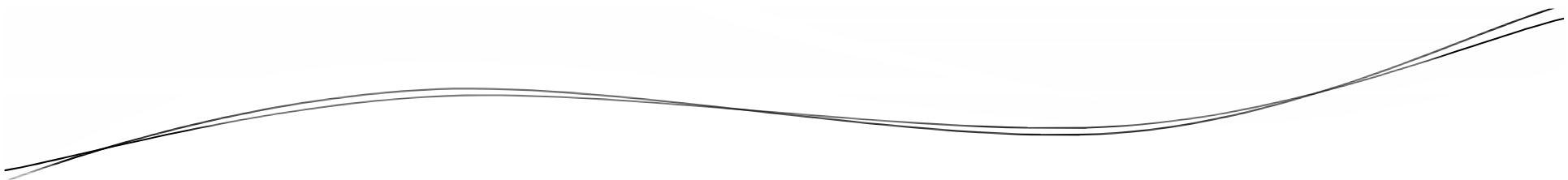
Why Inner Classes

- Inner classes are heavily used in a GUI programming model for handling events.
- They are also used for UI component creation.



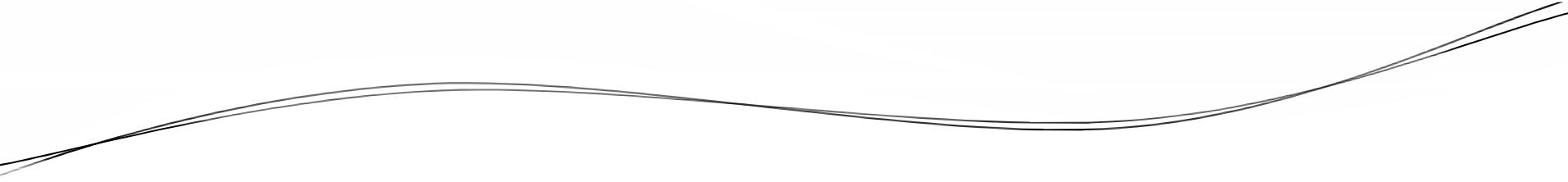
Types of Inner Classes

By Rahul Barve



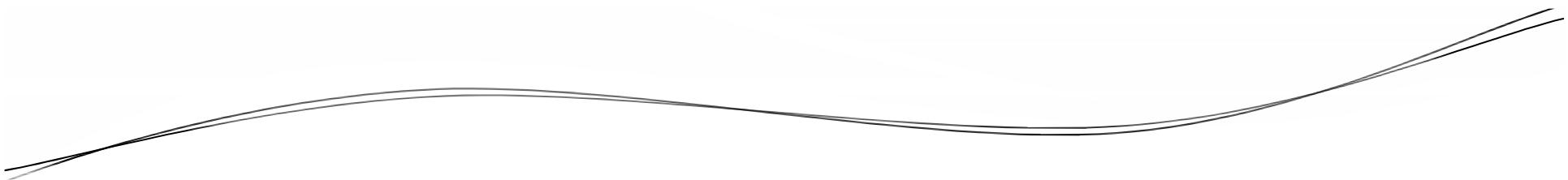
Types of Inner Classes

- Inner classes are divided into 4 types:
 - Static Inner Class
 - Nested Class
 - Local Class
 - Anonymous Inner Class



Static Inner Class

By Rahul Barve

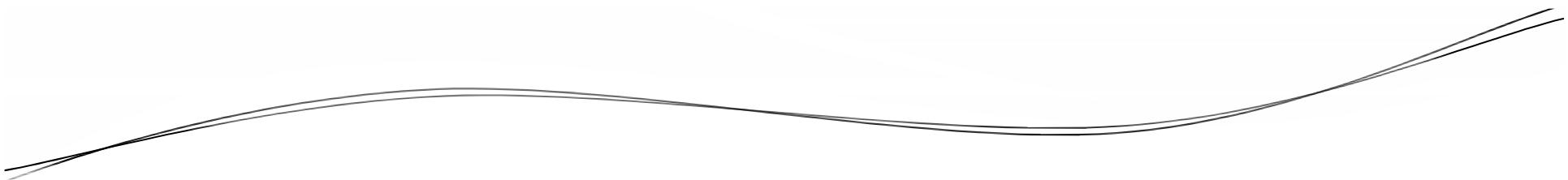


Static Inner Class

- An inner class declared with static modifier is called as a static inner class.
- It can be instantiated directly using the name of the outer class but can access only static members of an outer class.

Static Inner Class

```
public class OuterClass {  
    ... .  
    public static class InnerClass {  
        ... .  
    }  
}
```



Nested Class

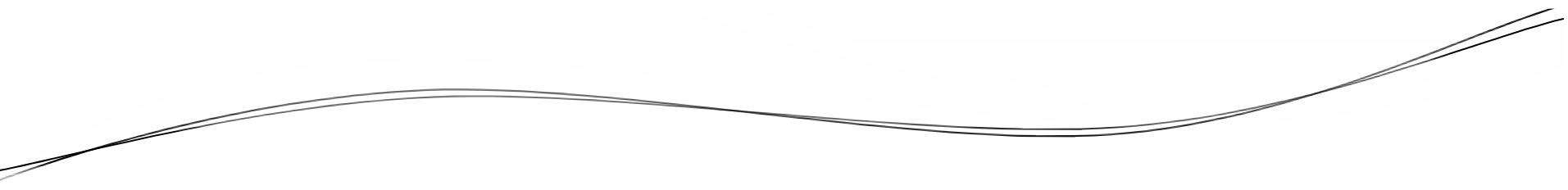
By Rahul Barve

Nested Class

- An inner class declared without static modifier is called as a nested class.
- It has to be instantiated always by using an object of an outer class but can access static as well as non-static members of an outer class.

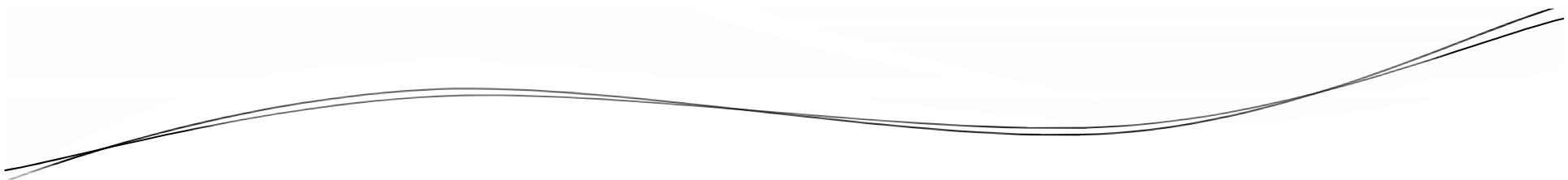
Nested Class

```
public class OuterClass {  
    .... .  
    public class InnerClass {  
        .... .  
    }  
}
```



Local Class

By Rahul Barve

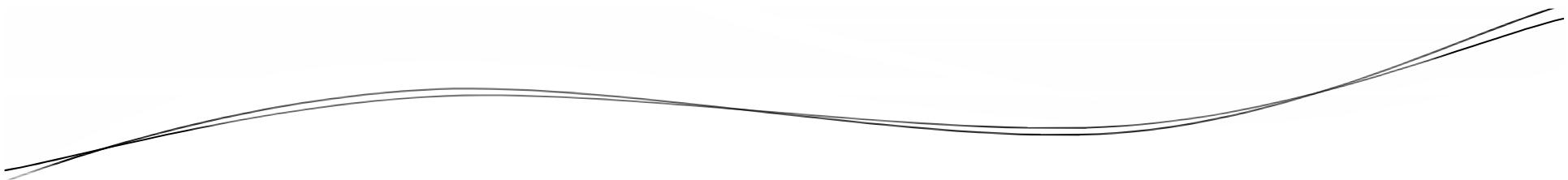


Local Class

- An inner class declared within a method's definition is known as a local class.
- It is loaded and instantiated for every method invocation.

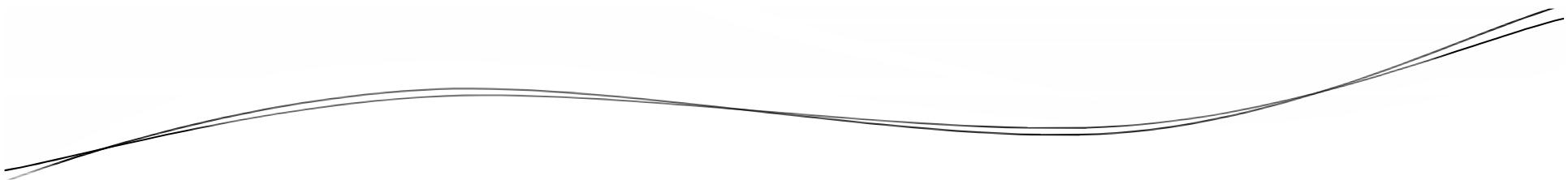
Local Class

```
public class OuterClass {  
    public void myMethod() {  
        class InnerClass {  
            ...  
        }  
    }  
}
```



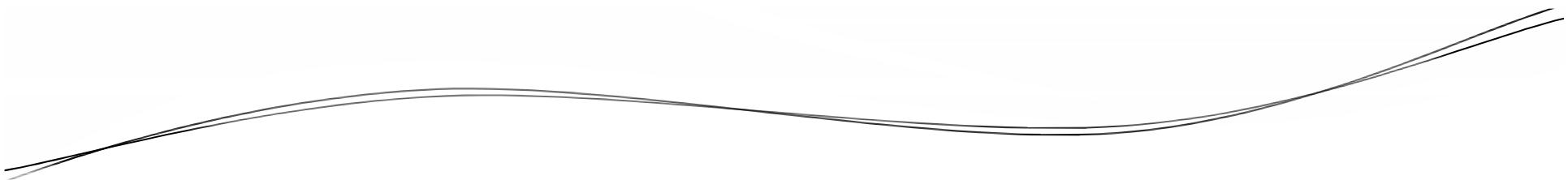
Anonymous Inner Class

By Rahul Barve



Anonymous Inner Class

- An inner class declared without any name is an anonymous inner class.
- It is defined, loaded and instantiated within the invocation of a method or a constructor.
- It is used in the context of abstract classes or interfaces.



Enums

By Rahul Barve

Enums

- An enum type is a special data type that enables for a variable to be a set of predefined constants.
- The variable must be equal to one of the values that have been predefined for it.

Enums

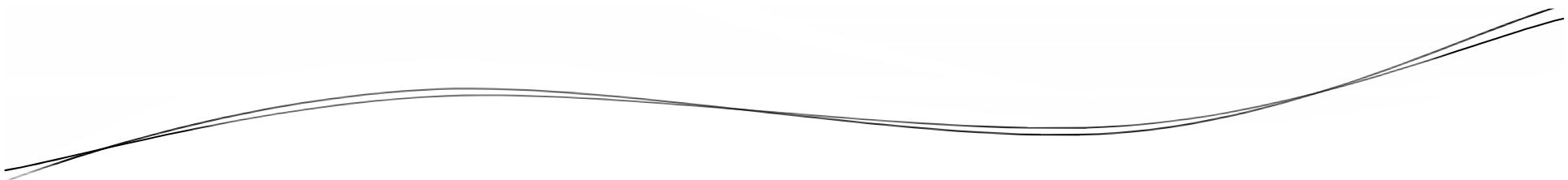
- Enums are declared using enum keyword.
- E.g.

```
public enum Nationality {  
    INDIAN,      US,      GERMAN,      BRITISH,  
    FRENCH,     JAPANESE,   OTHER  
}
```

Enums

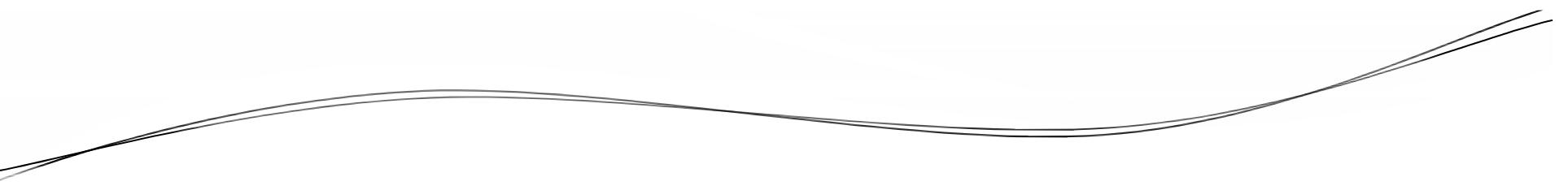
- Once an enum is declared, it can be used by using a dot (.) operator.
- E.g.

```
Nationality nt = Nationality.INDIAN;
```



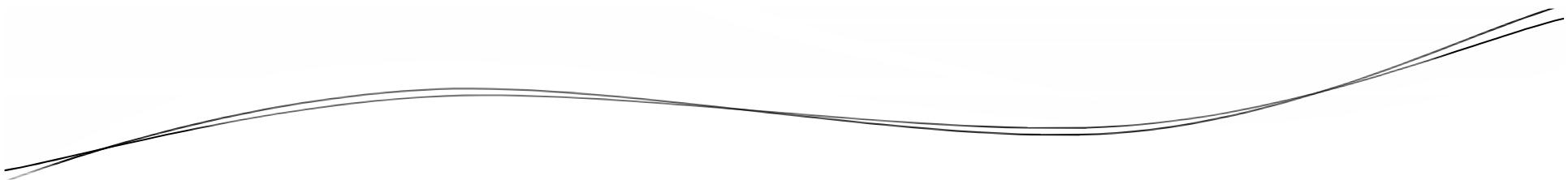
Enums

```
if (nt.equals(Nationality.INDIAN)) {  
    ...  
}
```



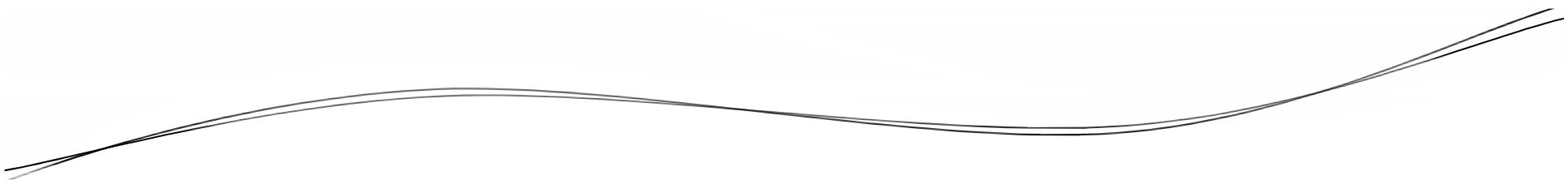
Reflection

By Rahul Barve



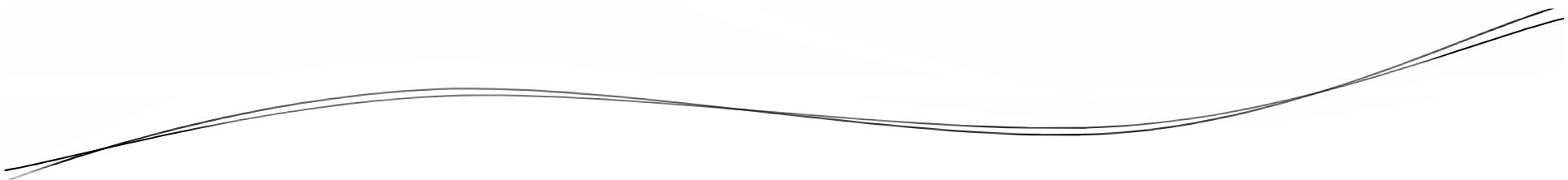
Reflection

- Sometimes, it's necessary to retrieve information about the class and perform some operations at runtime.
- Java provides a Reflection API that belongs to a package `java.lang.reflect`.



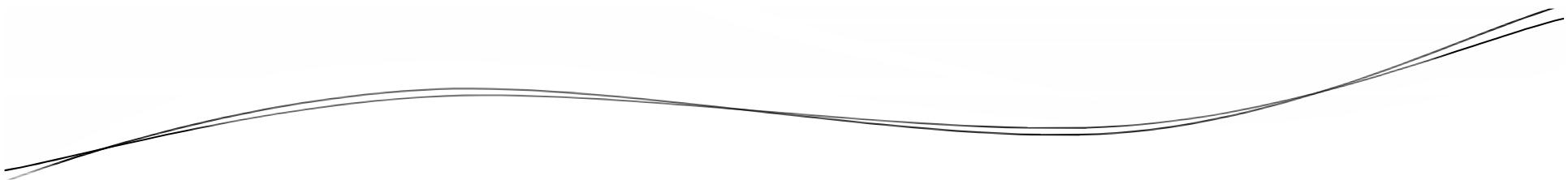
Reflection API

By Rahul Barve

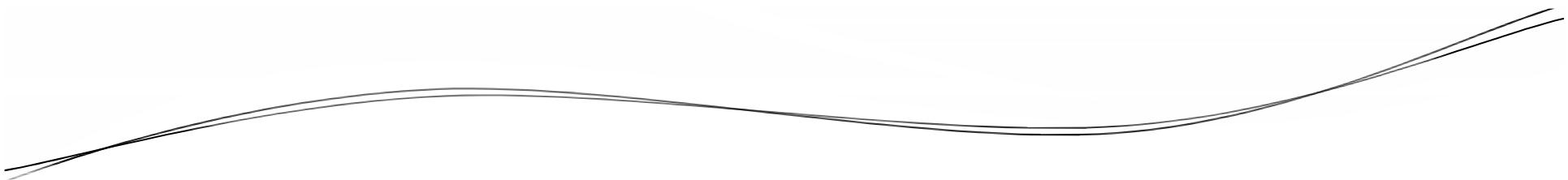


Reflection API

- Reflection API mainly consists of 4 classes:
 - `java.lang.Class`
 - `java.lang.reflect.Method`
 - `java.lang.reflect.Constructor`
 - `java.lang.reflect.Field`



Lambda Expressions

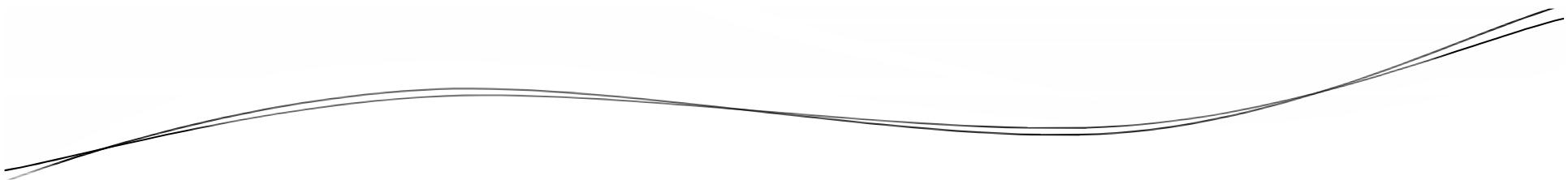


Lambda Expressions

- A lambda expression is a new syntax element and operator into the Java language.
- The operator `->` sometimes referred to as a *lambda operator* or an *arrow operator*.

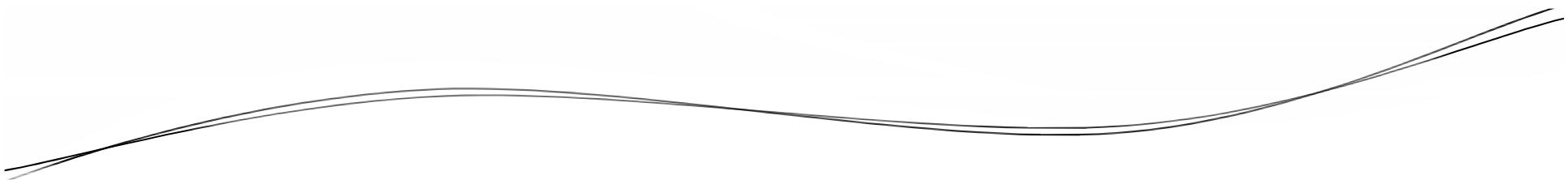
Lambda Expressions

- The lambda operator divides the expression into 2 parts.
- The left side indicates Lambda Parameters whereas the right side indicates Lambda Body.

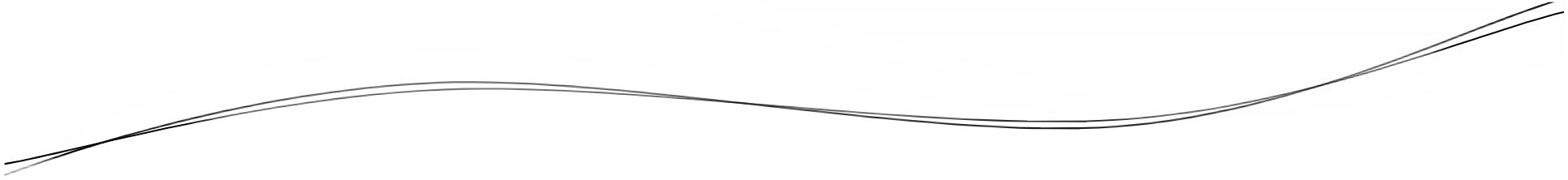


Lambda Expressions

- Lambda bodies are divided into 2 types:
 - Single Expression Lambda
 - Blocked Lambda

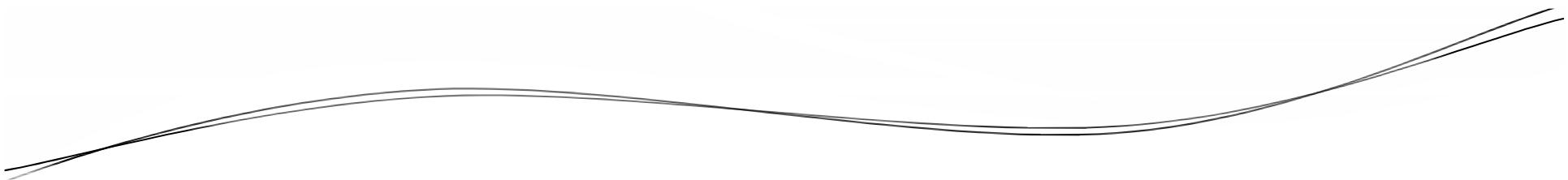


Date and Time API

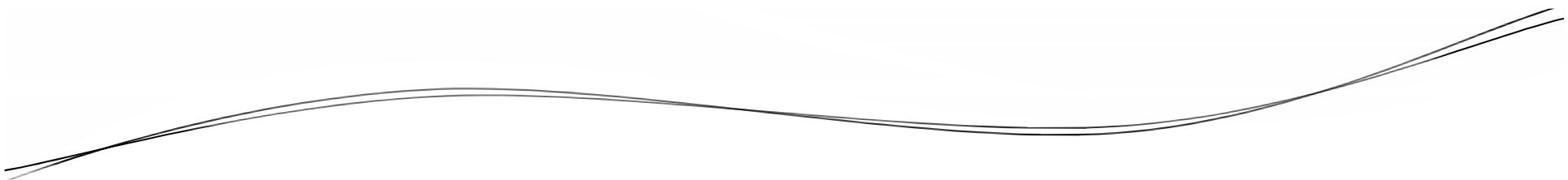


Date and Time API

- With the release of JDK 8, Java now includes another approach to handing time and date.
- The Date and Time API of JDK 8 simplifies processing of date and time.

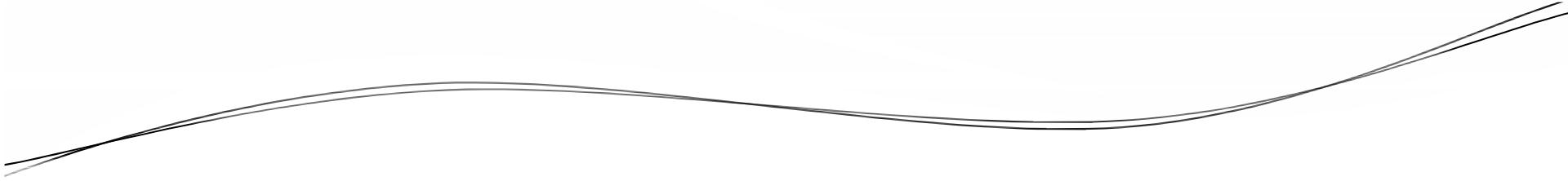


Why Date and Time API



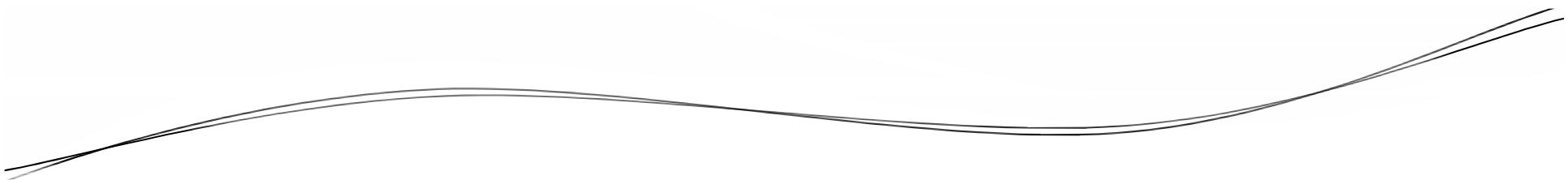
Why Date and Time API

- The existing classes aren't thread-safe, leading to potential concurrency issues for users.
- Some of the date and time classes also exhibit quite poor API design.



Why Date and Time API

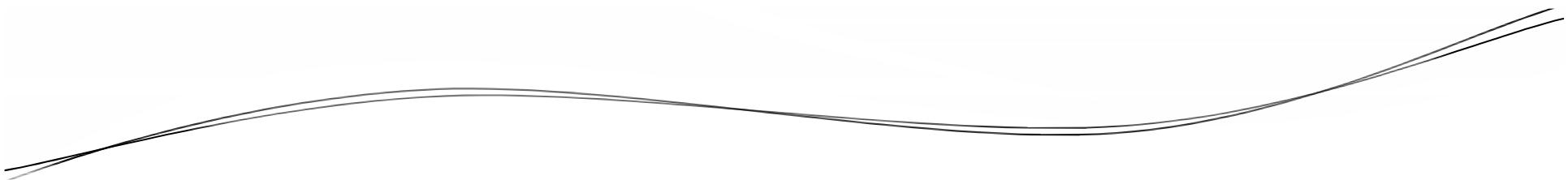
- For example, years in `java.util.Date` start at 1900, months start at 1, and days start at 0—not very intuitive.



Date and Time API

Date and Time API

- The `java.time` package is the heart of Date / Time API.
- It mainly consists of 3 classes:
 - `LocalDate`
 - `LocalTime`
 - `LocalDateTime`



Lets Summarize

- Inner Classes
- Using Enums
- Reflection API
- Lambda Expressions
- Date and Time API

LocalDateTimeExample.java ✘ CustomizedDateExample.java ✘

```
1 package language_essentials.date_and_time;
2
3 import java.time.LocalDate;
4 import java.time.LocalDateTime;
5 import java.time.LocalTime;
6
7 public class LocalDateTimeExample {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         LocalDate sysDate = LocalDate.now();
12         System.out.println("Today's date : " + sysDate);
13
14         LocalTime sysTime = LocalTime.now();
15         System.out.println("Current Time: " + sysTime);
16
17         LocalDateTime sysTimeStamp = LocalDateTime.now();
18         System.out.println("Current Time Stamp: " + sysTimeStamp);
19
20     }
21
22 }
23 }
```

LocalDateTimeExample.java CustomizedDateExample.java

```
1 package language_essentials.date_and_time;
2
3 import java.time.LocalDate;
4 import java.time.temporal.ChronoUnit;
5
6 public class CustomizedDateExample {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        LocalDate independenceDay = LocalDate.of(1947, 8, 15);
11        System.out.println("Independence Day: " + independenceDay);
12
13        LocalDate dateAfter2WeeksOfIndependence = independenceDay.plus(2, ChronoUnit.WEEKS);
14        System.out.println(dateAfter2WeeksOfIndependence);
15
16    }
17
18 }
19
```

A screenshot of a Java code editor interface. At the top, there are three tabs: "TrainingMode.java", "EnumExample.java", and "TrainingProgram.java". The "TrainingMode.java" tab is active, showing the following code:

```
1 package language_essentials.enums;
2
3 public enum TrainingMode {
4     OFFLINE, ONLINE, HYBRID
5 }
6
7
```

```
TrainingMode.java EnumExample.java TrainingProgram.java
1 package language_essentials.enums;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class EnumExample {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        List<TrainingProgram> trainingPrograms = new ArrayList<TrainingProgram>();
11        TrainingProgram program1 = new TrainingProgram();
12        TrainingProgram program2 = new TrainingProgram("Java Frameworks", TrainingMode.ONLINE);
13        TrainingProgram program3 = new TrainingProgram("DevOps", TrainingMode.ONLINE);
14        TrainingProgram program4 = new TrainingProgram("Mastering UI", TrainingMode.HYBRID);
15        TrainingProgram program5 = new TrainingProgram("Python and Django", TrainingMode.HYBRID);
16
17        trainingPrograms.add(program1);
18        trainingPrograms.add(program2);
19        trainingPrograms.add(program3);
20        trainingPrograms.add(program4);
21        trainingPrograms.add(program5);
22
23        showHybridTrainingPrograms(trainingPrograms);
24
25    }
26
27    private static void showHybridTrainingPrograms(List<TrainingProgram> trainingPrograms) {
28        // TODO Auto-generated method stub
29        for(TrainingProgram prg : trainingPrograms) {
30            TrainingMode currentTrainingMode = prg.getModeOfTraining();
31            if(currentTrainingMode.equals(TrainingMode.HYBRID))
32                System.out.println(prg);
33        }
34    }
35}
```

```
TrainingMode.java EnumExample.java TrainingProgram.java
25 }
26
27     private static void showHybridTrainingPrograms(List<TrainingProgram> trainingPrograms) {
28         // TODO Auto-generated method stub
29         for(TrainingProgram prg : trainingPrograms) {
30             TrainingMode currentTrainingMode = prg.getModeOfTraining();
31             if(currentTrainingMode.equals(TrainingMode.HYBRID))
32                 System.out.println(prg);
33         }
34     }
35
36 }
37
38 }
```

```
TrainingMode.java EnumExample.java TrainingProgram.java
1 package language_essentials.enums;
2
3 public class TrainingProgram {
4     private String name;
5     private TrainingMode modeOfTraining;
6     public TrainingProgram() {
7         name = "Java Full Stack";
8         modeOfTraining = TrainingMode.OFFLINE;
9     }
10    public TrainingProgram(String name, TrainingMode modeOfTraining) {
11        this.name = name;
12        this.modeOfTraining = modeOfTraining;
13    }
14    public String getName() {
15        return name;
16    }
17    public void setName(String name) {
18        this.name = name;
19    }
20    public TrainingMode getModeOfTraining() {
21        return modeOfTraining;
22    }
23    public void setModeOfTraining(TrainingMode modeOfTraining) {
24        this.modeOfTraining = modeOfTraining;
25    }
26    @Override
27    public String toString() {
28        return "TrainingProgram [name=" + name + ", modeOfTraining=" + modeOfTraining + "]";
29    }
30
31 }
32 }
```

Printable.java SimpleLambdaExpressionExample.java MathOperator.java MultipleParameterizedLambdaExpressionExample.java ParameterizedLambdaExpressionExample.java

```
1 package language_essentials.lambda;
2 @FunctionalInterface
3 public interface Printable {
4     void print();
5 }
6
```

```
Printable.java x SimpleLambdaExpressionExample.java x MathOperator.java x MultipleParameterizedLambdaExpressionExample.java x ParameterizedLambdaExpressionExample.java x
1 package language_essentials.lambda;
2
3 public class SimpleLambdaExpressionExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7
8         //Single Expression Lambda
9         Printable prn = () -> System.out.println("Welcome");
10        Printable prn2 = () -> System.out.println("Hello");
11
12        //Blocked Lambda
13        Printable prn3 = () -> {
14            System.out.println("Good Morning");
15            System.out.println("Good Afternoon");
16        };
17
18        prn.print();
19        prn2.print();
20        prn3.print();
21
22    }
23
24 }
25
```

Printable.java SimpleLambdaExpressionExample.java MathOperator.java MultipleParameterizedLambdaExpressionExample.java ParameterizedLambdaExpressionExample.java

```
1 package language_essentials.lambda;
2
3 public interface MathOperator {
4     int getResult(int a, int b);
5 }
6
```

Printable.java SimpleLambdaExpressionExample.java MathOperator.java MultipleParameterizedLambdaExpressionExample.java ParameterizedLambdaExpressionExample.java

```
1 package language_essentials.lambda;
2
3 public class MultipleParameterizedLambdaExpressionExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         MathOperator opAdd = (x, y) -> x + y;
8         MathOperator opSubtract = (int a, int b) -> a - b;
9
10        int result = opAdd.getResult(10, 20);
11        System.out.println(result);
12
13        result = opSubtract.getResult(60, 45);
14        System.out.println(result);
15    }
16
17
18 }
19
```

```
Printable.java X SimpleLambdaExpressionExample.java X MathOperator.java X MultipleParameterizedLambdaExpressionExample.java X ParameterizedLambdaExpressionExample.java X
1 package language_essentials.lambda;
2
3 import interfaces.CurrencyConverter;
4
5 public class ParameterizedLambdaExpressionExample {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         CurrencyConverter euroToRupeesConverter = amt -> amt * 90.45f;
10        //When a method returns a value, do not use 'return' statement while working with single expression lambda
11        float inr = euroToRupeesConverter.doConvert(1000);
12        System.out.println(inr);
13
14        CurrencyConverter yenToRupeesConverter = (amountInYen) -> {
15            float rupees = amountInYen / 1.7f;
16            return rupees;
17        };
18        inr = yenToRupeesConverter.doConvert(20000);
19        System.out.println(inr);
20
21    }
22
23 }
24
```

```
1 package language_essentials.reflection;
2
3 import java.lang.reflect.Field;
4 import java.lang.reflect.Method;
5
6 import collections.Movie;
7
8 public class ReflectionExample {
9     private static void createObject(String className) throws ClassNotFoundException, InstantiationException, IllegalAccessException {
10         Class cls = Class.forName(className); //Loads the class explicitly
11         System.out.println("Class loaded");
12         cls.newInstance(); //Creates an instance of the class that is loaded
13     }
14     private static void printClassInfo(Object obj) {
15         Class currentClass = obj.getClass();
16         String className = currentClass.getName();
17         System.out.println("Class name: " + className);
18         System.out.println("-----Printing names of the methods-----");
19         Method methods[] = currentClass.getDeclaredMethods();
20         for(Method mt : methods) {
21             String methodName = mt.getName();
22             System.out.println(methodName);
23         }
24
25         System.out.println("-----Printing names of the fields-----");
26         Field fields[] = currentClass.getDeclaredFields();
27         for(Field f : fields) {
28             String fieldName = f.getName();
29             System.out.println(fieldName);
30         }
31     }
}
```

```
Printable.java X SimpleLambdaExpressionExample.java X MathOperator.java X MultipleParameterizedLambdaExpressionExample.java X ParameterizedLambdaExpressionExample.java X ReflectionExample.java X
22     System.out.println(methodName);
23 }
24
25 System.out.println("-----Printing names of the fields-----");
26 Field fields[] = currentClass.getDeclaredFields();
27 for(Field f : fields) {
28     String fieldName = f.getName();
29     System.out.println(fieldName);
30 }
31 }
32
33 public static void main(String[] args) {
34     // TODO Auto-generated method stub
35     //printClassInfo(new Movie());
36     try {
37         createObject("collections.Movie");
38     } catch (Exception e) {
39         // TODO Auto-generated catch block
40         e.printStackTrace();
41     }
42 }
43 }
44 }
45 }
46 }
```