# Title: Java Basics

## Objectives:

- Understand the different types of constructors.
- Understand reference type and value type.
- Use Access Specifiers to limit the scope of member attributes and methods.
- Get familiar with Getter and Setters.
- Exception Handling

## Theory:

- **Types of constructors:** constructor is a special method which runs whenever a object is instantiated from a class. They are decalred without a return type and are always public. One class can have more than one constructor. There are three types of constructors:

  1. **Default constructor:** This type of constructor is defined without any parameters.

```
class MyClass{
    int a;
    MyClass(){
        this.a = 5;
    }
}
```

  2. **Parameterized constructor:** This type of constructor are defined with one or many parameters.

```
class MyClass{
    int a;
    MyClass(int a){
        this.a = a;
    }
}
```

  3. **Copy constructor:** This type of constructor are defined with a parameter which accepts the object of the same class.

```
class MyClass{
    int a;
    MyClass(MyClass M){
        this.a = m.a;
    }
}
```

- **Reference types:** When arguments are passed to a method they are either passed as value or reference. In C/C++ the programmer manually sets wether to pass a reference or value. But in java it is determined automatically by the type of argument. When argument is passed as refernce the pointer to the object is passed and changes to inner scope is also applied to outer scope. When argument is passed as value a copy is made so the changes of inner scope is not seen in outer scope. In java the primitive types are int,float,char and boolean and reference types are ClassTypes,ArrayTypes or TypeVariable.

- **Setters and Getters:** In OOP we can set visibility of fields and methods to be only accessible from the class or accessible from anywhere. In java there are two visibility public, visible anywhere, private, visible only inside the class. TO access the private members we use getters and setters. They must be public and by convention are declared as getVariable and setVariable.

```
class MyClass{
    private int a;
    public int getA(){
        return this.a;
    }
    public void setA(int a){
        this.a = a;
    }
}
```

- **Exception handling:** Exception are runtime errors. They need to be handled properly for code safety. `try..catch..final` expression is used for exception handling in Java. The code that can throw an Exception is wrapped by the try block, when a run time error occurs it thrown an instance of Exception class or a class which extends the Exception class. The catch block is executed if an exception occurs. The finally block is optional it runs in any condition.

```
try {
    c = a / b;
} catch (ArithmeticException e) {
System.out.println("Division error " + e);
} finally {
    System.out.println("c = " + c);
}
```

**Program 1: Program to demonstrate that default constructor is created by compiler if no constructors are explicitly defined.**

```
public class DefaultConstructor {
    int a;
    public static void main(String[] args) {
        DefaultConstructor c = new DefaultConstructor();
        //equivalent program in c will give randoum value
        System.out.println(c.a);
    }
}
```

**Output:** 0

**Program 2: Program to test default constructor is not created automatically if any other constructor is defined explicitly.**

```
public class NoDefault {
    int a;
    NoDefault(int a){
        this.a = a;
    }
    public static void main(String[] args) {
        //Error: The constructor NoDefault() is undefined
        NoDefault nd = new NoDefault();
    }
}
```

**Output:**

```
    mdhe@mdheKoLaptop:~/coding/java/lab2$ javac NoDefault.java
    NoDefault.java:8: error: constructor NoDefault in class NoDefault cannot be applied to given types;
            NoDefault nd = new NoDefault();
                           ^
        required: int
        found:    no arguments
        reason: actual and formal argument lists differ in length
    1 error
```

**Program 3: Program to demonstrate the use of all types of constructors.**

```
class Student {
    int roll;
    String name;

    // parameterized constructor
    Student(int roll, String name) {
        this.roll = roll;
        this.name = name;
    }
    // default constructor
    Student() {
        roll = 0;
        name = "";
    }
    // Copy constructor
    Student(Student S) {
        roll = S.roll;
        name = S.name;
    }
    public String toString() {
        return "Roll = " + this.roll + " Name = " + this.name;
    }
}

public class AllConstructors {
```

```
    public static void main(String[] args) {
        Student S1 = new Student();
        Student S2 = new Student(4,"hari");
        Student S3 = new Student(S2);
        System.out.println(S1);
        System.out.println(S2);
        System.out.println(S3);
    }
}
```

**Output:**

```
Roll = 0 Name =
Roll = 4 Name = hari
Roll = 4 Name = hari
```

**Program 4: Program to demonstrate reference type with copy constructor:**

```
class Room{
    float area;
    boolean copied;
    Room(float area){
        this.area = area;
        copied = false;
        area = 0;
    }
    Room(Room R){
        this.area = R.area;
        R.copied = true;
    }
    public String toString() {
        return "Area = " + area + " copied = " + copied;
    }
}


public class CopyConstructor {
    public static void main(String[] args) {
        float roomArea =  50.56f;
        //roomArea is passed as value
        Room room1 = new Room(roomArea);
        //No change in the roomArea variable
        System.out.println(roomArea);
        //room1 is passed as reference
        Room room2 = new Room(room1);
        System.out.println("Room 1 ");
        System.out.println(room1);
        System.out.println("Room 2");
        System.out.println(room2);
    }
}
```

**Output:**

```
50.56
Room 1
Area = 50.56 copied = true
Room 2
Area = 50.56 copied = false
```

**Program 5: Program to demonstrate the use of access specifier and how getter and setter helps to access the private attributes.**

```
import java.util.Scanner;

class Account {
    private String username;
```

```
    private String password;
    private static int accountCount = 0;
    public int id;
    Account(){
        id = accountCount;
        accountCount++;
    }
    public boolean setUsername(String username) {
        if (username.contains("BadWord")) {
            System.out.println("Do not include bad words");
            return false;
        }
        this.username = username;
        return true;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public boolean setPassword(String password) {
        if (password.length() < 8) {
            System.out.println("Minimum password length is 8");
            return false;
        }
        this.password = password;
        return true;
    }

}

public class AccessSpecifier {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        Account newAccount = new Account();
        boolean success = false;
        String line = "";
        while (!success) {
            System.out.println("Enter username");
            line = in.next();
            success = newAccount.setUsername(line);
        }
        success = false;
        while (!success) {
            System.out.println("Enter password");
            line = in.next();
            success = newAccount.setPassword(line);
        }
        System.out.println("id = " + newAccount.id + " username = " + newAccount.getUsername() + " password
        in.close();
    }
}
```

**Output:**

```
Enter username
MyBadWord
Do not include bad words
Enter username
ayush
Enter password
```

```
abcd
Minimum password length is 8
Enter password
abcdefghij
id = 0 username = ayush password = abcdefghij
```

**Program 6: Program to demonstrate the use of exception handling**

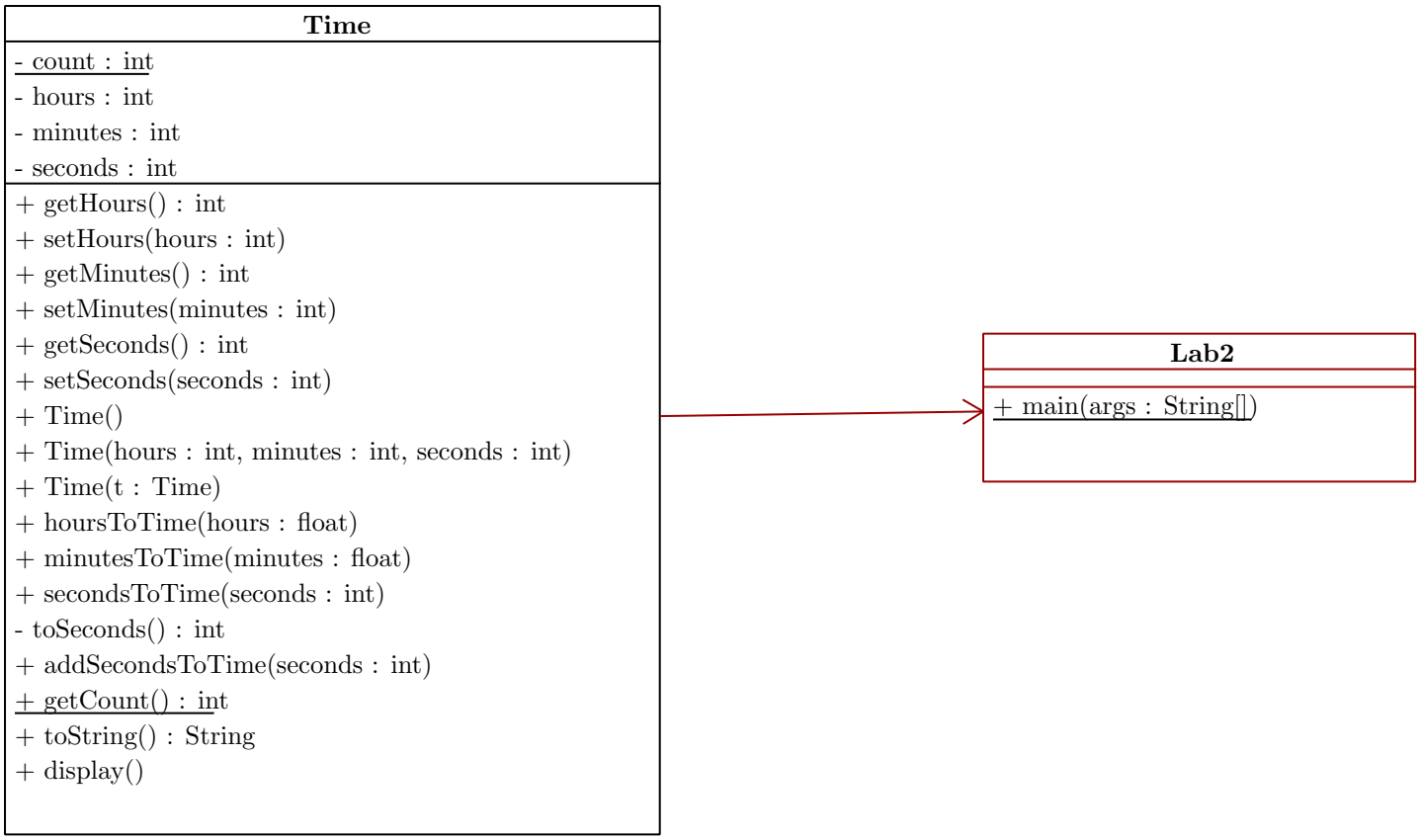
```
import java.util.Scanner;

public class ExceptionHandling {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int a = 0;
        int b = 0;
        String line = "";
        System.out.println("Enter value for a");
        line = in.next();
        try {
            a = Integer.parseInt(line);
        } catch (NumberFormatException e) {
            System.out.println("Unhandled input " + e);
            System.out.println("Set a as 0");
        }
        System.out.println("Enter value for b");
        line = in.next();
        try {
            b = Integer.parseInt(line);
        } catch (NumberFormatException e) {
            System.out.println("Unhandled input " + e);
            System.out.println("Set b as 0");
        }
        float c = 0;
        try {
            c = a / b;
        } catch (ArithmeticException e) {
           System.out.println(e.getMessage());
        } finally {
            System.out.println("c = " + c);
        }
        in.close();
    }
}
```

**Output:**

```
Enter value for a
5
Enter value for b
e
Unhandled input java.lang.NumberFormatException: For input string: "e"
Set b as 0
/ by zero
c = 0.0
```

**Program 7: Program to creeate a Time class with various attributes and methods**




```java
class InvalidValueException extends Exception {
    public InvalidValueException(String s) {
        super(s);
    }
}

class Time {
    static private int count = 0;
    private int hours;

    private int minutes;

    private int seconds;

    public int getHours() {
        return hours;
    }

    public void setHours(int hour) throws InvalidValueException {
        if (hour >= 0) {
            this.hours = hour;
        } else {
            throw new InvalidValueException("Hour should be > 0");
        }
    }

    public int getMinutes() {
        return minutes;
    }

    public void setMinutes(int minute) throws InvalidValueException {
        if (minute >= 0 && minute < 60) {
            this.minutes = minute;
        } else {
            throw new InvalidValueException("Minute should be > 0");
        }
    }
```

```
    public int getSeconds() {
        return seconds;
    }

    public void setSeconds(int seconds) throws InvalidValueException {
        if (seconds >= 0 && seconds < 60) {
            this.seconds = seconds;
        } else {
            throw new InvalidValueException("Second should be > 0");
        }
    }

    public Time(Time t) {
        try {
            setHours(t.hours);
            setMinutes(t.minutes);
            setSeconds(t.seconds);
            count++;
        } catch (InvalidValueException e) {
            System.out.println(e.getMessage());
        }
    }

    public Time(int hour, int minute, int second) {
        try {
            setHours(hour);
            setMinutes(minute);
            setSeconds(second);
            count++;
        } catch (InvalidValueException e) {
            System.out.println(e.getMessage());
        }
    }

    public Time() {
        hours = 0;
        minutes = 0;
        seconds = 0;
        count++;
    }

    public void hoursToTime(float hours) throws InvalidValueException {
        secondsToTime((int) (hours * 3600));
    }

    public void minutesToTime(float mins) throws InvalidValueException {
        secondsToTime((int) (mins * 60));
    }

    public void secondsToTime(int seconds) throws InvalidValueException {
        float h = (seconds / 3600f);
        setHours((int) h);
        float m = (h - hours) * 60;
        setMinutes((int) m);
        float s = (m - minutes) * 60;
        setSeconds((int) s);
    }

    int toSeconds() {
        return hours * 3600 + minutes * 60 + seconds;
    }

    public void addSecondsToTime(int seconds) throws InvalidValueException {
```

```
        secondsToTime(toSeconds() + seconds);
    }

    public static int getCount() {
        return count;
    }

    public String toString() {
        String retString = "";
        retString = retString + (hours < 10 ? ("0" + hours) : hours) + ":";
        retString = retString + (minutes < 10 ? ("0" + minutes) : minutes) + ":";
        retString = retString + (seconds < 10 ? ("0" + seconds) : seconds);
        return retString;
    }

    public void display() {
        System.out.println(this);
    }
}

public class Lab2 {
    public static void main(String[] args) throws Exception {
        Time t1 = new Time();
        t1.setHours(15);
        t1.setMinutes(5);
        t1.setSeconds(7);
        t1.display();
        // demonstrating parameterized constructor
        Time t2 = new Time(15, 5, 7);
        t2.display();
        // demonstrating copy constructor
        Time t3 = new Time(t1);
        t3.addSecondsToTime(53);
        t3.display();
        // generating exception
        try {
            t3.setSeconds(-5);
        } catch (InvalidValueException e) {
            System.out.println(e.getMessage());
        } finally {
            System.out.println("Finally called");
        }
        Time t4 = new Time();
        t4.hoursToTime(5.87f);
        t4.display();
        Time t5 = new Time();
        t5.minutesToTime(100.5f);
        t5.display();
        Time t6 = new Time();
        t6.secondsToTime(82600);
        t6.display();
        System.out.println(Time.getCount());
    }
}
```

**Output:**

```
15:05:07
15:05:07
15:06:00
Second should be > 0
Finally called
05:52:11
01:40:29
00:08:20
```

6

**Conclusion:** We learned about different types of constructors, refernce types, copy constructors, Access Specifiers and Error handling.