Dr. Ambedkar Institute of Technology, Bengaluru

Department of Computer Science & Engineering

UG Programme

Subject Name: JAVA Programming
Subject Code: 18CS52

Presentation

By

Dr. Smitha Shekar B
Associate Professor
Dept.of CSE, Dr.A.I.T

Dr. Smitha Shekar B

UNIT-2 CONTENTS

INHERITANCE

EXCEPTION HANDLING

UNIT 2 - TOPICS THAT WILL BE COVERED

- Inheritance
  - Basics
  - Multilevel Hierarchy
- Exception Handling
  - Exception-Handling Fundamentals
  - Exception Types

INHERITANCE IN JAVA

Dr. Smitha Shekar B

INHERITANCE IN JAVA

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- It is an important part of OOPs (Object Oriented programming system).
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- When you inherit from an existing class, you can reuse methods and fields of the parent class.
- You can add new methods and fields in your current class also.
- Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Inheritance in Java

Why use inheritance in java?

1. For Method Overriding (so runtime polymorphism can be achieved).
2. For Code Reusability.

Terms used in Inheritance

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Inheritance in Java

The syntax of Java Inheritance

```
class Subclass-
name extends Superclass-name
{
   //methods and fields
}
```

The **extends** keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

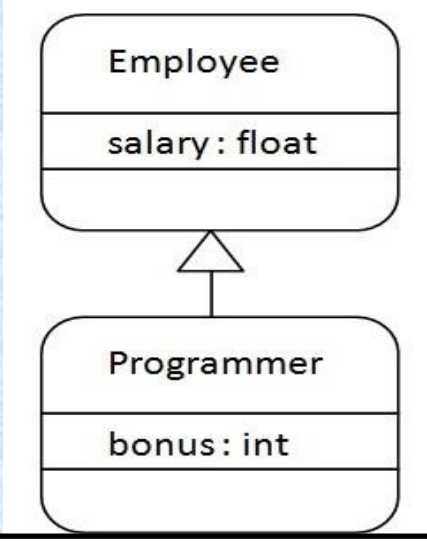
Inheritance in Java

Java Inheritance Example

As displayed in the above figure, Programmer is the subclass and Employee is the superclass.
The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);

   System.out.println("Bonus of Programmer is:"+p.bonu
s);
}
}
```



OUTPUT:

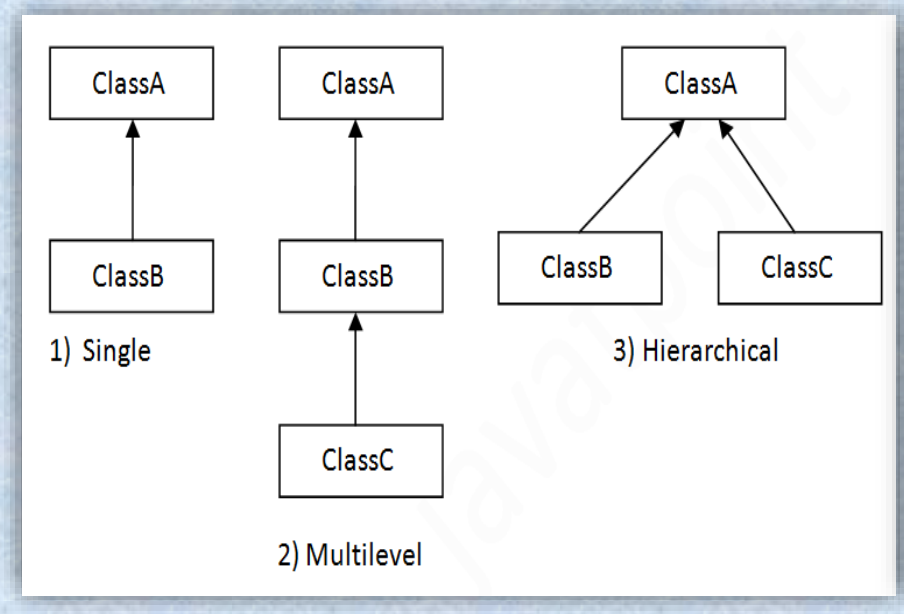
Programmer salary is:40000.0
Bonus of programmer is:10000

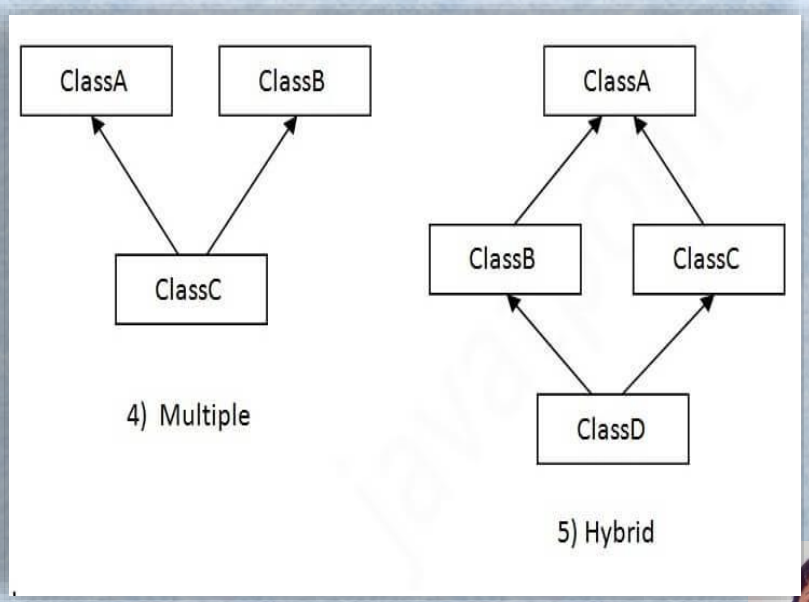
Inheritance in Java

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.





Inheritance in Java

Single Inheritance Example

When a class inherits another class, it is known as a single inheritance.

In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

File: TestInheritance.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:

barking...
eating...

Inheritance in Java

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as multilevel inheritance.

As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

File: TestInheritance2.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

weeping...
barking...
eating...

Inheritance in Java

Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as hierarchical inheritance.
In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

File: TestInheritance3.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

meowing...
eating...

Inheritance in Java

Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invok
ed?
}
}
```

Output:

Compile Time Error

Inheritance in Java

Aggregation in Java

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Example: Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

```
class Employee{
int id;
String name;
Address address;//Address is a class
...
}
```

Employee has an entity reference address, so relationship is Employee HAS-A address.

Why use Aggregation?

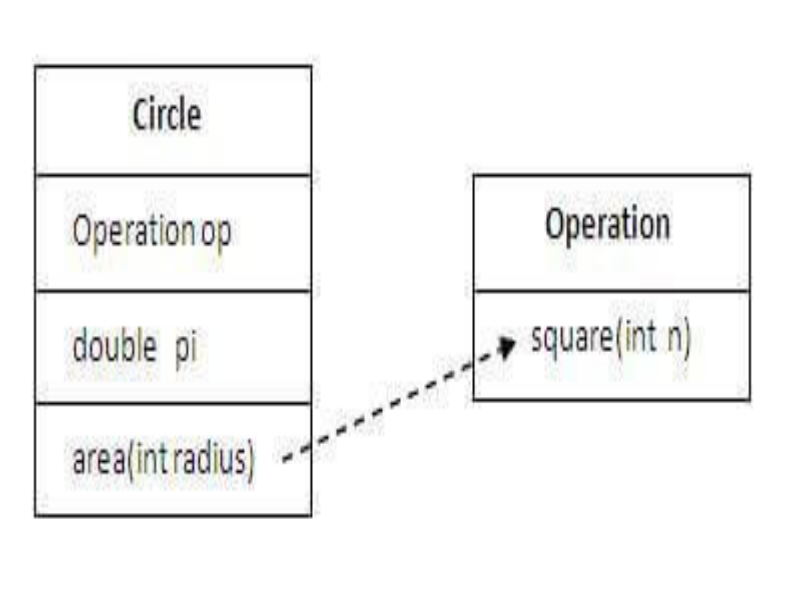
- For Code Reusability.

```
class Operation{
 int square(int n){
  return n*n;
 }
}

class Circle{
 Operation op;//aggregation
 double pi=3.14;

 double area(int radius){
   op=new Operation();
   int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).

   return pi*rsquare;
 }
```



Output:78.5

we have created the reference of Operation class in the Circle class.

Dr. Smitha Shekar B

When use Aggregation?

- Code reuse is also best achieved by aggregation when there is no IS-A relationship.
- Inheritance should be used only if the relationship IS-A is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Dr. Smitha Shekar B

Example

- Employee has an object of Address, address object contains its own informations such as city, state, country etc.
- In such case relationship is, Employee HAS-A address.

Address.java

```
class Address
{
 String city, state, country;
 Address(String city, String state, String country)
   {
      this.city = city;
      this.state = state;
      this.country = country;
    }

}
```

Emp.java

```
public class Emp {
int id;
String name;
Address address;

Emp(int id, String name, Address address)
{
   this.id = id;
   this.name = name;
   this.address=address;
}

void display(){
System.out.println(id+" "+name);
System.out.println(address.city+" "+addres
s.state+" "+address.country);
}
```

```
public static void main(String[] args)
{
Address address1=new Address("Blr
","Karnataka","India");
Address address2=new Address("My
s","Karnataka","India");

Emp e=new Emp(111,"Smitha",addr
ess1);
Emp e2=new Emp(112,"Savitha",add
ress2);

e.display();
e2.display();

}
}
```


Java Polymorphism

- Method Overloading
- Method Overriding

Dr. Smitha Shekar B

Method Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
- In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Dr. Smitha Shekar B

Usage of Java Method Overriding

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

Rules for Java Method Overriding

The method must have the same name as in the parent class

The method must have the same parameter as in the parent class.

There must be an IS-A relationship (inheritance).

Dr. Smitha Shekar B

```
//Java Program to demonstrate why we need method overriding

//Here, we are calling the method of parent class with child
//class object.
//Creating a parent class
class Vehicle{
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike extends Vehicle{
  public static void main(String args[]){
  //creating an instance of child class
  Bike obj = new Bike();
  //calling the method with child class instance
  obj.run();
  }
}
```

Dr. Smitha Shekar B

```
//Java Program to illustrate the use of Java Method Overriding

//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");}

  public static void main(String args[]){
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

Bike is running safely

Dr. Smitha Shekar B

NOTE

we have defined the run method in the subclass as defined in the parent class but it has some specific implementation.

The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

Dr. Smitha Shekar B

Example

```
class TestMethodOverride
 {
    void show(String a)
     {
      System.out.println(“1”);
     }
 }
class  ABC extends TestMethodOverride
{
    void show(String a, int b)
     {
       System.out.println(“2”);
      }
    public static void main(String[] args)
      {
         TestMethodOverride t = new TestMethodOverride();
          t.show(“Hi”);
          t.show(“Hello”,20);
       }
}
```

Dr. Smitha Shekar B

EXCEPTION - HANDLING IN JAVA

Dr. Smitha Shekar B

Check List

- Exception
- Hierarchy
- Exception handling mechanisms
- Keywords
- Checked and Unchecked Exceptions
- Sample programs

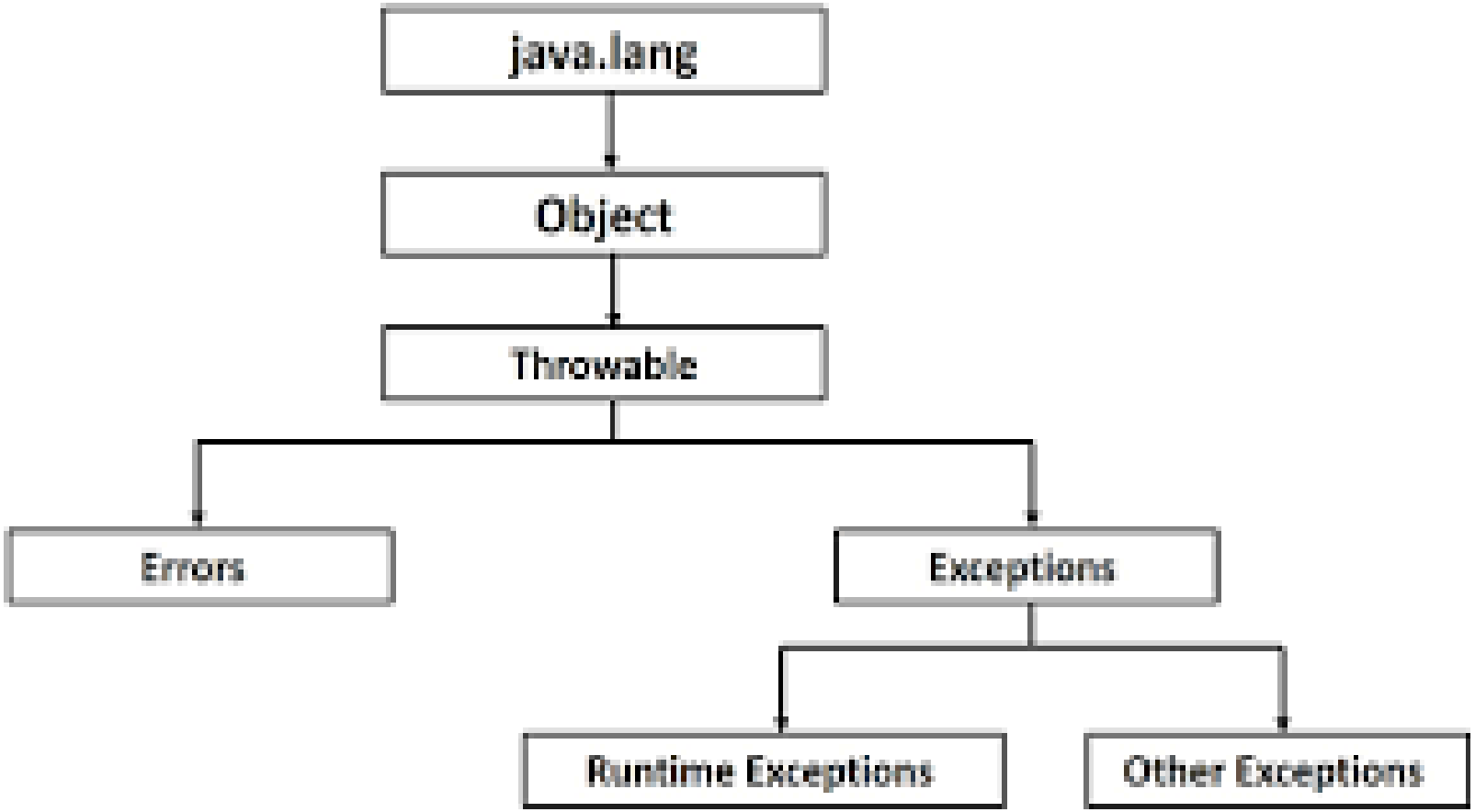
Dr. Smitha Shekar B

Exception Handling in JAVA

- An exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e at runtime, that disrupts the normal flow of the program.
- An exception is an abnormal condition that arises in a code sequence at run time.
- In other words, an exception is a run-time error.

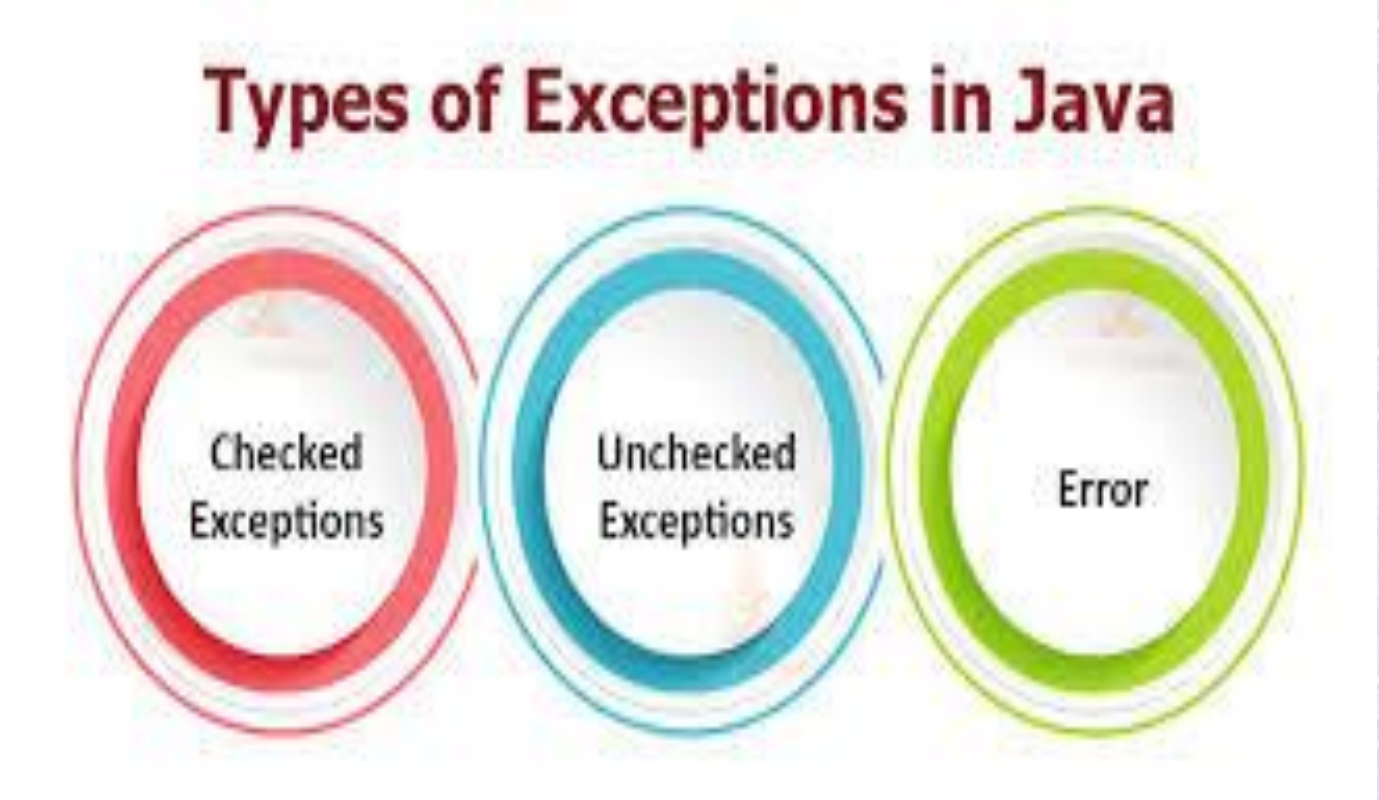
Dr. Smitha Shekar B

Hierarchy of Exception Classes



Dr. Smitha Shekar B

Exception Types



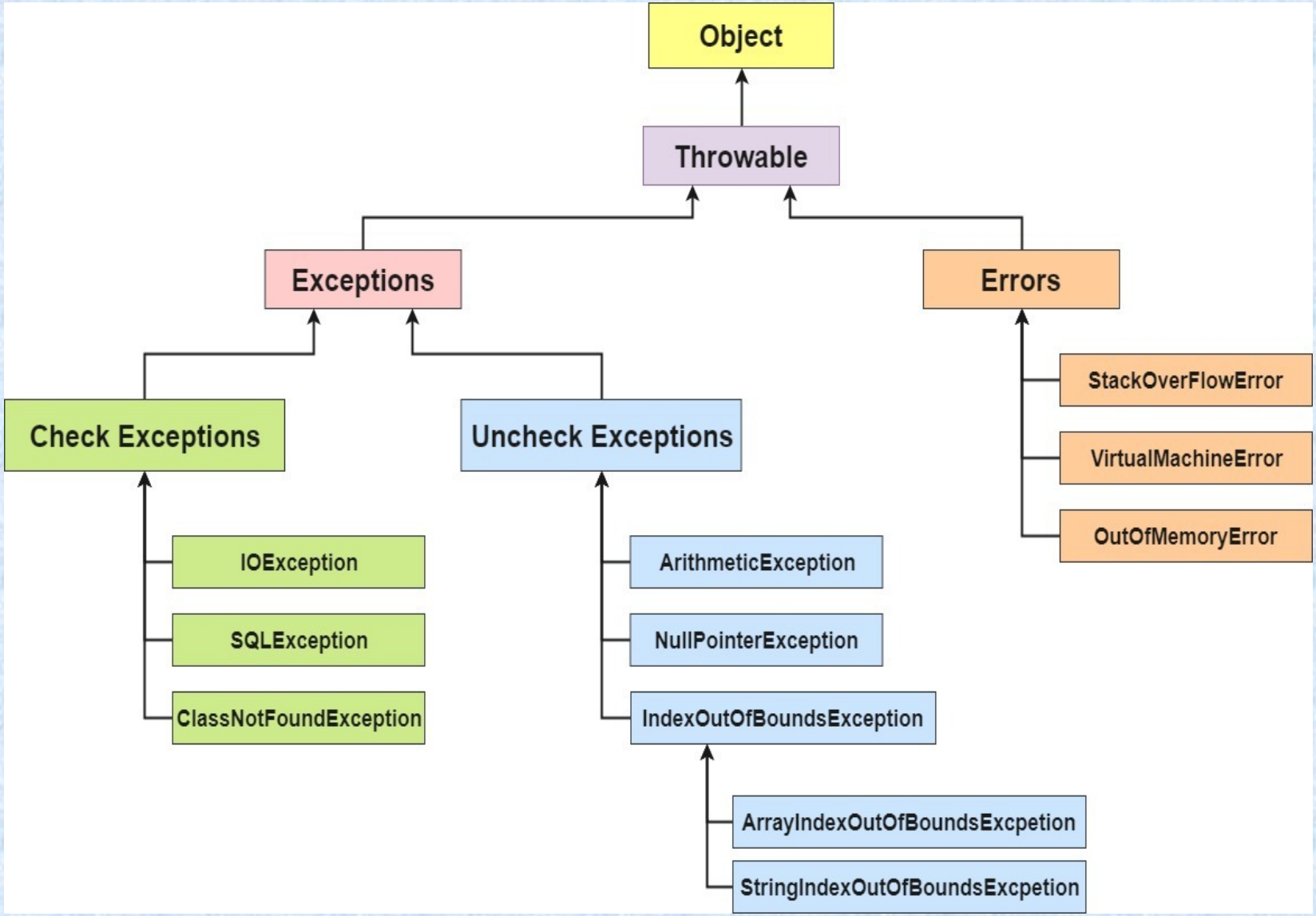
Dr. Smitha Shekar B

Difference between Error and Exception

- **Errors** indicate that something severe enough has gone wrong, the application should crash rather than try to handle the error.

- **Exceptions** are events that occurs in the code. A programmer can handle such conditions and take necessary corrective actions.

Dr. Smitha Shekar B



Dr. Smitha Shekar B

Checked vs. Unchecked Exceptions

Checked Exceptions	Unchecked Exceptions
Not subclass of RuntimeException	Subclass of RuntimeException
if not caught, method *must* specify it to be thrown	if not caught, method *may* specify it to be thrown
for errors that the programmer *cannot* directly prevent from occurring	For errors that the programmer *can* directly prevent from occurring
`IOException`, `FileNotFoundException`, `SocketException`, etc.	`NullPointerException`, `IllegalArgumentException`, `IllegalStateException`, etc.

Dr. Smitha Shekar B

Few examples

- NullPointerException – When you try to use a reference that points to null.
- ArithmeticException – When bad data is provided by user, for example, when you try to divide a number by zero this exception occurs because dividing a number by zero is undefined.
- ArrayIndexOutOfBoundsException – When you try to access the elements of an array out of its bounds, for example array size is 5 (which means it has five elements) and you are trying to access the 10th element.

Dr. Smitha Shekar B

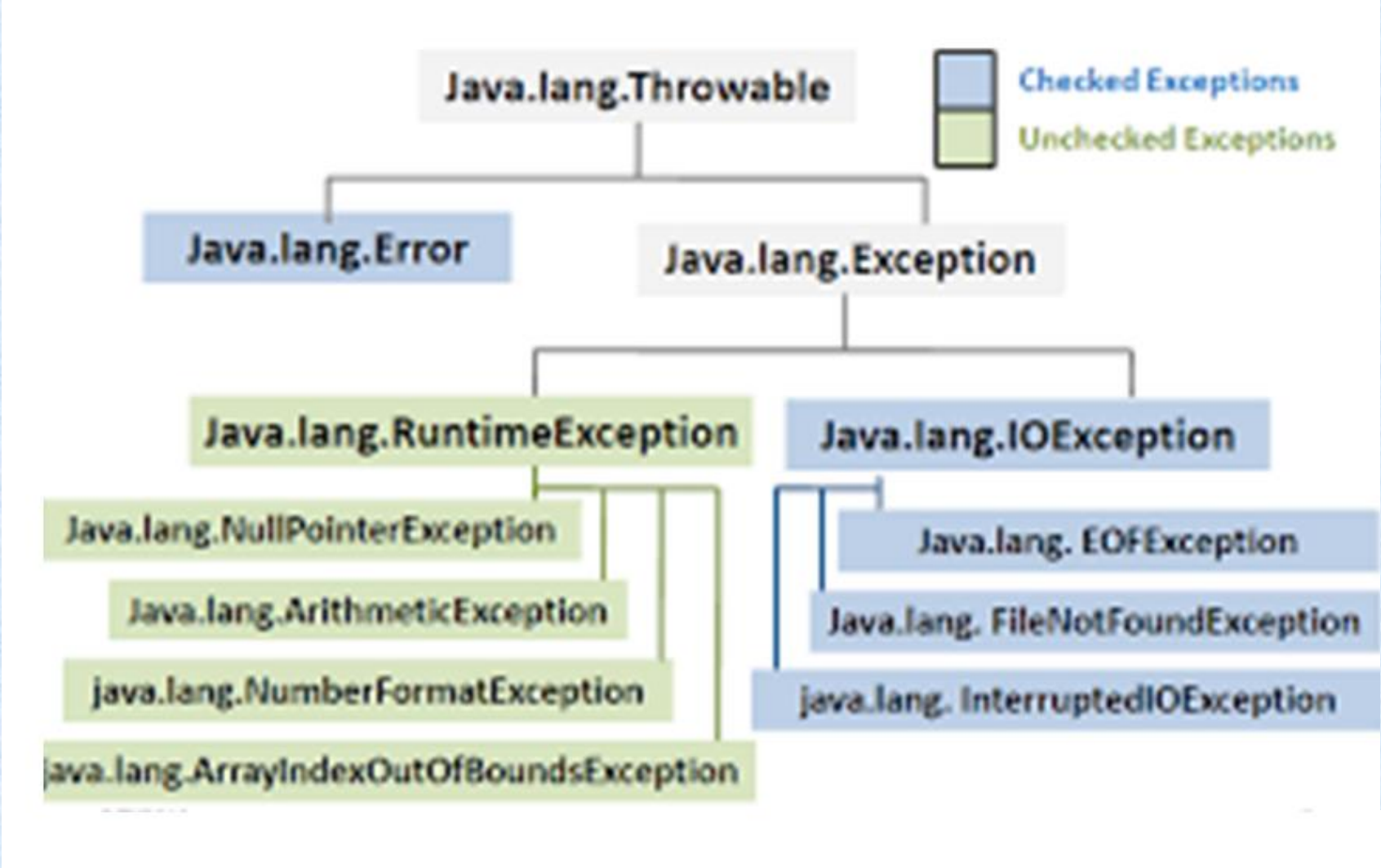
Advantage of Exception Handling

- Exception handling ensures that the flow of the program doesn't break when an exception occurs.
- For example, if a program has bunch of statements and an exception occurs mid way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly.

- By handling we make sure that all the statements execute and the flow of program doesn't break.

Dr. Smitha Shekar B

Hierarchy of Exception Classes



Dr. Smitha Shekar B

Types of exceptions

- There are two types of exceptions in Java:
  1)Checked exceptions
  2)Unchecked exceptions

1) Checked exceptions

- All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not.
- If these exceptions are not handled/declared in the program, you will get compilation error.
- For example, SQLException, IOException, ClassNotFoundException etc.

Dr. Smitha Shekar B

2) Unchecked Exceptions

- Runtime Exceptions are also known as Unchecked Exceptions.
- These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit.
- For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.
- Compiler will never force you to catch such exception or force you to declare it in the method using throws keyword.

Dr. Smitha Shekar B

Keywords used for Exception Handling

- In Java it is handled using 5 keywords
  - try
  - catch
  - throw
  - throws
  - finally

- try – A block of source code that is to be monitored for exception
- catch – It handles the specific type of exception along with the try block.
- throw – It is used to throw specific exception from the program code.
- throws – It specifies the exception that can be thrown by a particular method.
- finally – it specifies the code that must be executed even though exception may or may not be occur
- Note:
  - For ever try block there exist the catch block.

Thank You

Dr. Smitha Shekar B