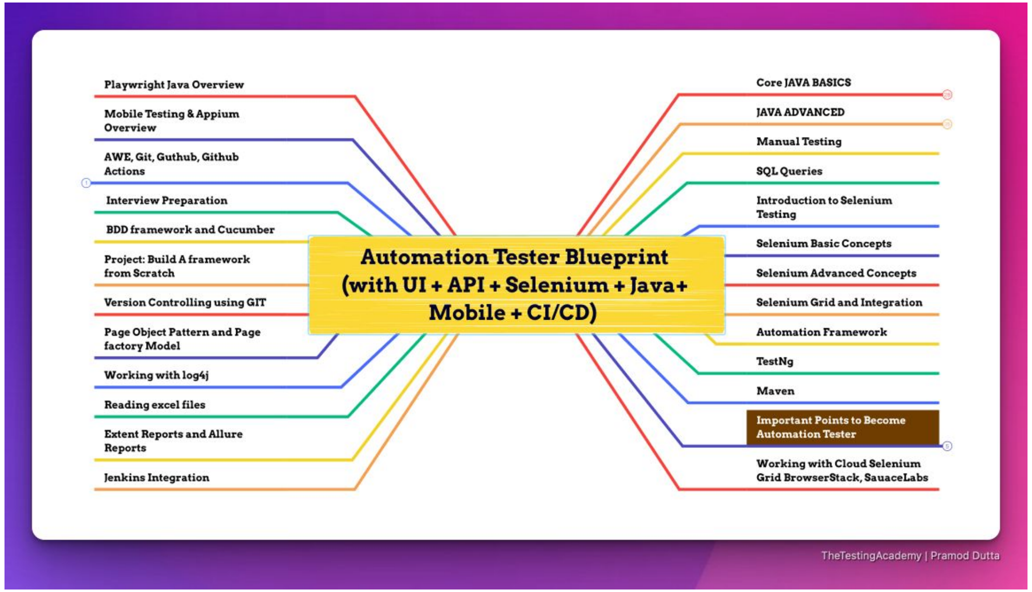
Meeting Etiquette

- Please be on Mute all the time.
- Turn off Video, So that we can save bandwidth.
- There is separate Q&A Section in End Please use that, Add questions in Google Form.
- Break at 5 min.
- Mute your microphone
- To help keep background noise to a minimum, make sure you mute your microphone when you are not speaking.
- Be mindful of background noise
- Avoid multitasking
- All links and Slides will be shared.

{ Automation Tester } BluePrint.

Pramod Dutta



Automation Tester Blueprint (with UI + API + Selenium + Java+ Mobile + CI/CD)
Playwright Java Overview
Mobile Testing & Appium Overview
AWE, Git, Guthub, Github Actions
Interview Preparation
BDD framework and Cucumber
Project: Build A framework from Scratch
Version Controlling using GIT
Page Object Pattern and Page factory Model
Working with log4j
Reading excel files
Extent Reports and Allure Reports
Jenkins Integration
Core JAVA BASICS
JAVA ADVANCED
Manual Testing
SQL Queries
Introduction to Selenium Testing
Selenium Basic Concepts
Selenium Advanced Concepts
Selenium Grid and Integration
Automation Framework
TestNg
Maven
Important Points to Become Automation Tester
Working with Cloud Selenium Grid BrowserStack, SauaceLabs
TheTestingAcademy | Pramod Dutta

Core Java + Interview Tips

Core Java & OOPs

Agenda

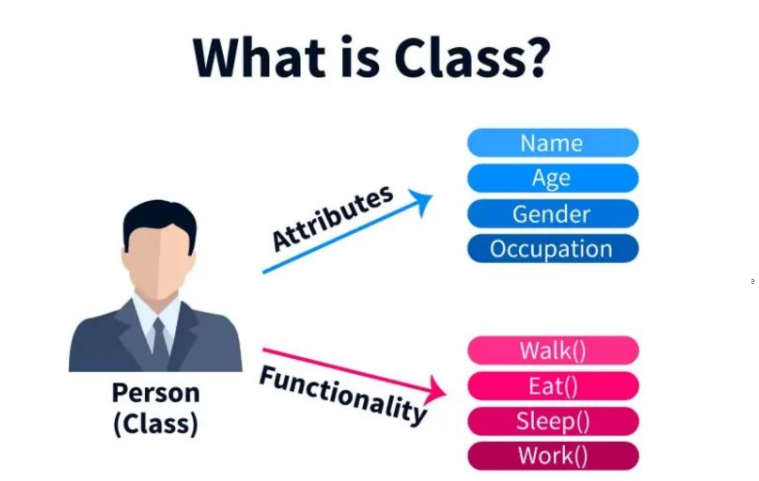
- Loops, Switch, do While, While
- Constructor
  a. Non-Parameterized constructor
  b. Parameterized constructor
  c. Copy Constructor
- Polymorphism
  a. Compile Time Polymorphism
  b. Runtime Polymorphism

Agenda

- Inheritance
- Package in Java
- Access Modifiers in Java
- Encapsulation
- Abstraction
- Interfaces
- Static Keyword
- Learning Maven Project
- Maven Lifecycle & Phases , Plugins

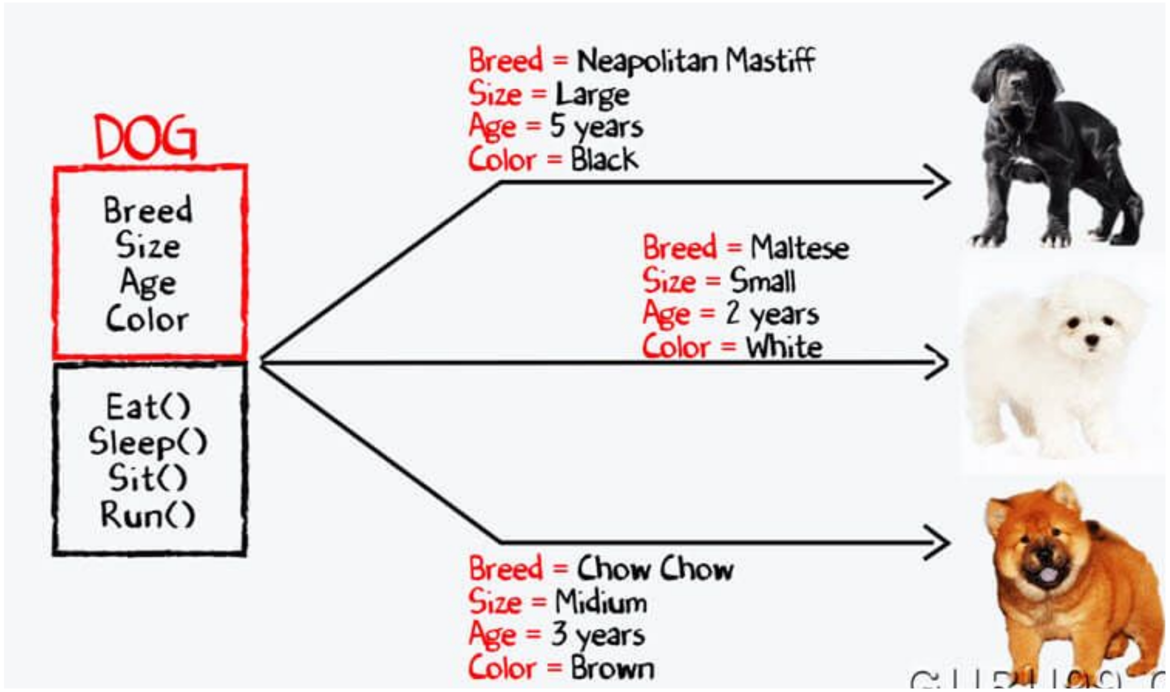
Class

A class is a group of objects which have common properties. A class can have some properties and functions (called methods).
The class we have used is Main.



When a class is created, no memory is allocated

Objects

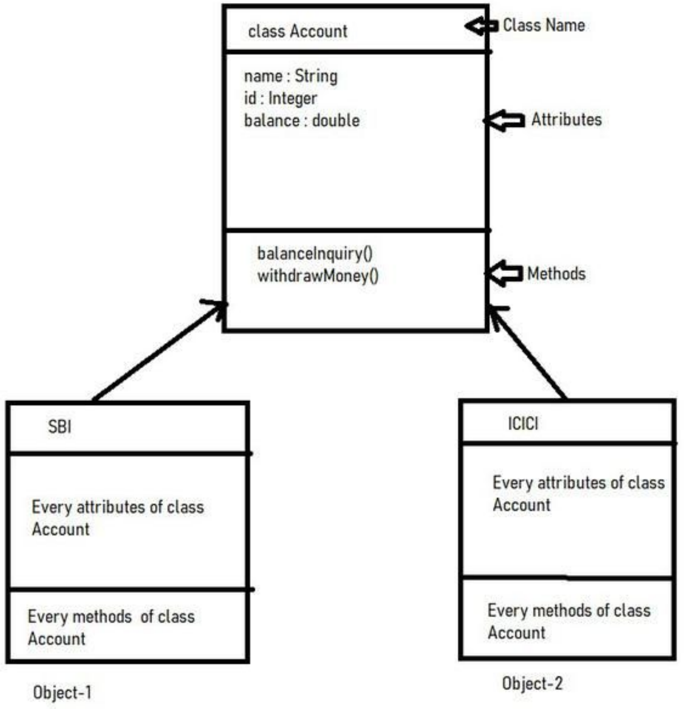


Object is an instance of a class. All data members and member functions of the class can be accessed with the help of objects

401
Response

Objects are allocated memory space whenever they are created.

Class & Objects



401
Response

Class

Every class in Java can be composed of the following elements:

- `fields, member variables or instance variables` - These are variables that hold data specific to a particular object. For each object of a class, there is one field.
- `member methods or instance methods` - These perform operations on objects and are defined within the class.
- `static or class fields` - These fields are common to any object within the same class. They can only exist once in the class irrespective of the total number of objects in the class.
- `static or class methods` - These methods do not affect a specific object in the class.
- `inner classes` - At times a class will be defined within a class if it is a subset of another class. The class contained within another is the **inner** class.

OBJECT ORIENTED PROGRAMMING

Object-Oriented Programming is a **methodology or paradigm** to design a program using classes and objects.

It simplifies the software development and maintenance by providing some concepts

Class is a user-defined data type which defines its properties and its functions

401
Response

Object is a run-time entity. It is an instance of the class. An object can represent a person, place or any other item

OBJECT ORIENTED PROGRAMMING

‘this’ keyword : ‘this’ keyword in Java that refers to the current instance of the class

In OOPS it is used to:
pass the current object as a parameter to another method
refer to the current class instance variable

401
Response

Constructor

Constructor is a **special method** which is invoked automatically at the time of object creation.

It is used to **initialize the data members** of new objects generally.
Constructors have the same name as class or structure.
Constructors don't have a return type. (Not even void)
Constructors are only called once, at object creation.

401
Response

There can be **three types** of constructors in Java.

Non-Parameterized constructor : A constructor which has no argument is known as non-parameterized constructor(or no-argument constructor). It is invoked at the time of creating an object. If we don't create one then it is created by default by Java.

Constructor

Parameterized constructor : Constructor which has parameters is called a parameterized constructor. It is used to provide different values to distinct objects.

Copy Constructor : A Copy constructor is an **overloaded** constructor used to declare and initialize an object from another object. There is only a user defined copy constructor in Java(C++ has a default one too).

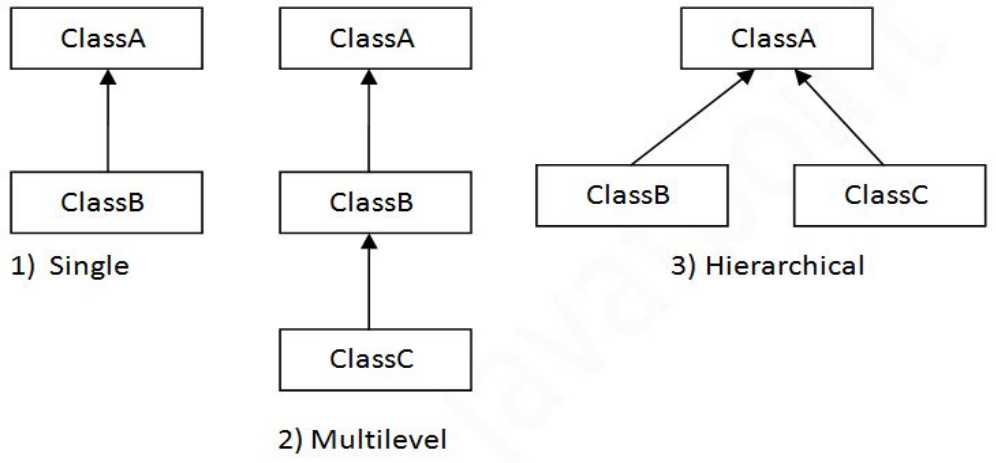
401
Response

Inheritance

Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.

Single inheritance

Hierarchical inheritance



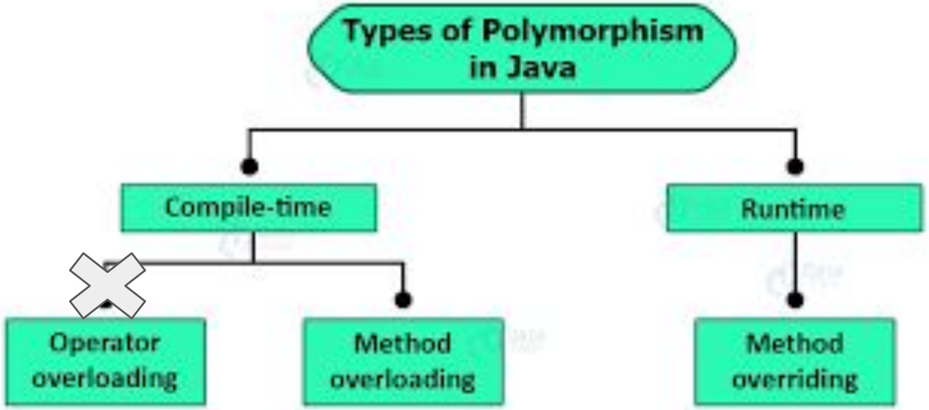
401
Response

Polymorphism

Polymorphism is the ability of an object to take on many forms

Each of these classes will have different underlying data. Precisely, Poly means ‘many’ and morphism means ‘forms’.

when a parent class reference is used to refer to a child class object



Polymorphism

Compile Time Polymorphism – The polymorphism which is implemented at the compile time is known as compile-time polymorphism

Method Overloading

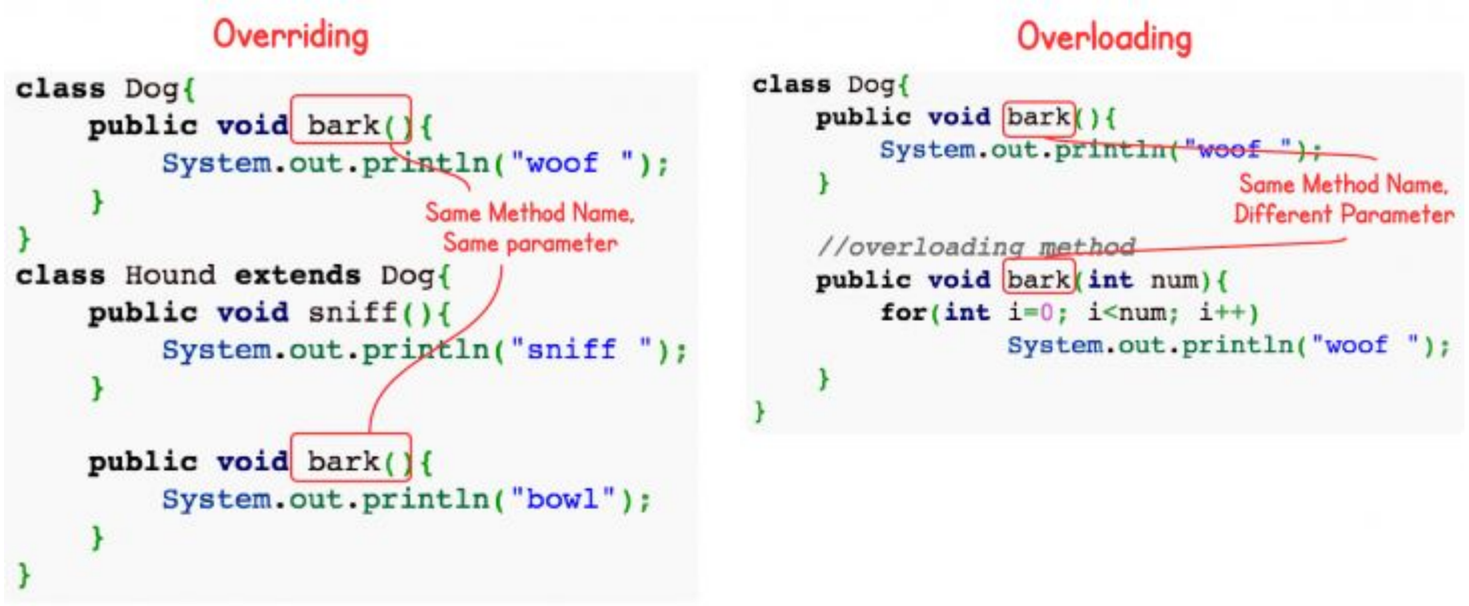
401
Response

Runtime Polymorphism

Runtime polymorphism is also known as **dynamic polymorphism**.
Function overriding is an example of runtime polymorphism.

Function overriding

Polymorphism

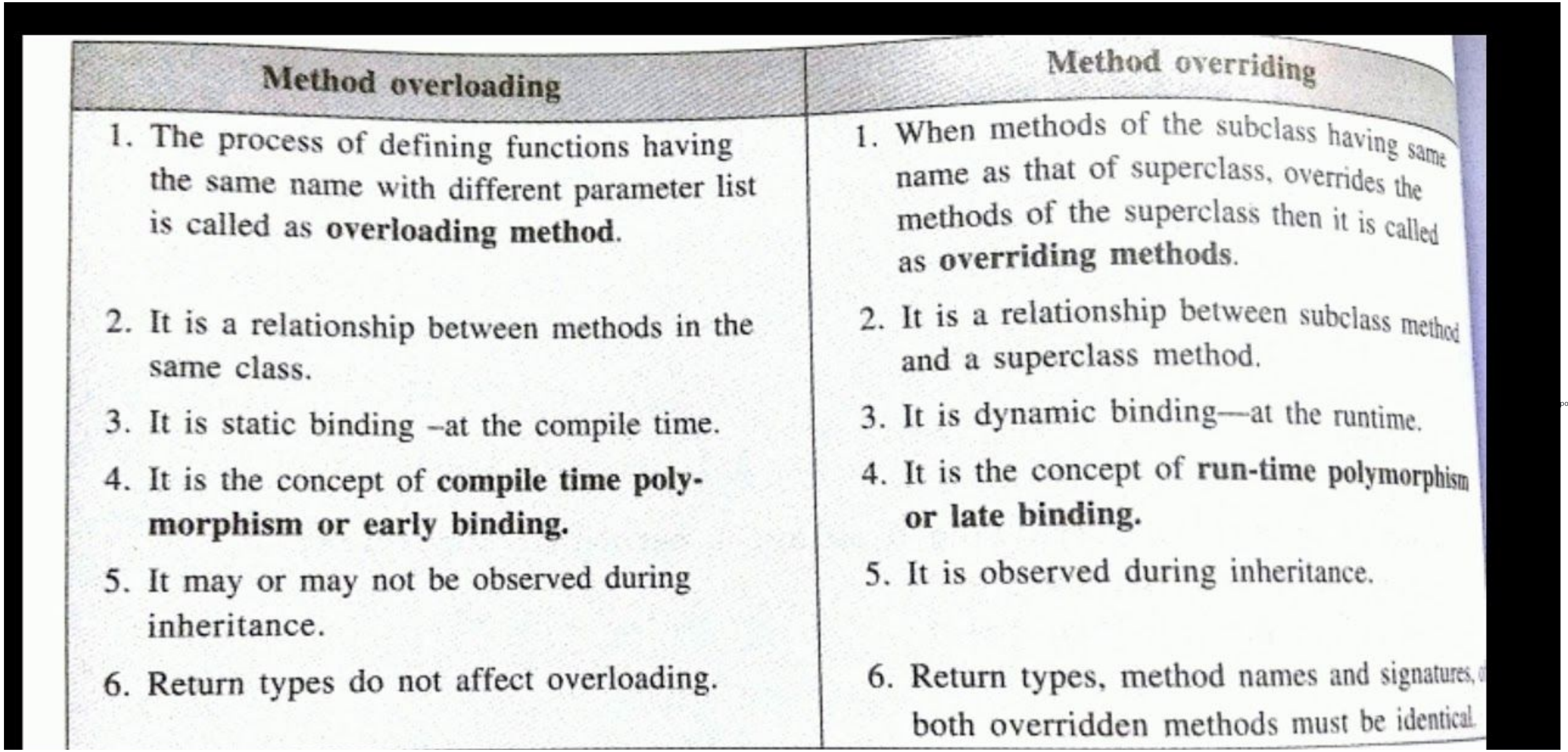

Overriding
Same Method Name, Same parameter
Overloading
Same Method Name, Different Parameter

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

401
Response

Polymorphism



Method overloading	Method overriding
1. The process of defining functions having the same name with different parameter list is called as **overloading method**.	1. When methods of the subclass having same name as that of superclass, overrides the methods of the superclass then it is called as **overriding methods**.
2. It is a relationship between methods in the same class.	2. It is a relationship between subclass method and a superclass method.
3. It is static binding –at the compile time.	3. It is dynamic binding—at the runtime.
4. It is the concept of **compile time poly-morphism or early binding.**	4. It is the concept of **run-time polymorphism or late binding.**
5. It may or may not be observed during inheritance.	5. It is observed during inheritance.
6. Return types do not affect overloading.	6. Return types, method names and signatures, o both overridden methods must be identical.

Assignment

1. Create the Class of ATB
2. Create an Array of ATB Students and add toString method.
3. Create Single, Multilevel, and Heriichca. Inteheri
4. Create both Overloading and OverRIDING Examples

Package in Java

Package is a group of similar types of classes, interfaces and sub-packages.
Packages can be built-in or user defined.

Built-in packages - java,
util, io etc.

import java.util.Scanner;

import java.io.IOException;

401
Response

Access Modifiers in Java

Private: The access level of a private modifier is only **within the class**. It cannot be accessed from outside the class.

Default: The access level of a default modifier is only **within the package**. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

401
Response

Protected: The access level of a protected modifier is **within the package** and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Abstraction

Abstraction is the quality of dealing with ideas rather than events.

when you consider the case of e-mail, complex details such as what happens as soon as you send an e-mail, the protocol your e-mail server uses are hidden from the user.

401
Response

Abstraction is achieved in 2 ways :
Abstract class
Interfaces (Pure Abstraction)

Abstraction

Demo

```
public abstract class Employee {
   private String name;
   private String address;
   private int number;

   public abstract double computePay();
   // Remainder of class definition
}
```

401
Response

Abstraction

```
abstract class Animal {
   abstract void walk();
   void breathe()
```

Abstract Methods

```
public abstract class Employee {
   private String name;
   private String address;
   private int number;

   public abstract double computePay();
   // Remainder of class definition
}
```

401
Response

Abstraction

Abstract Class

An abstract class must be declared with an abstract keyword.
It can have abstract and non-abstract methods.
It cannot be instantiated.
It can have constructors and static methods also.
It can have final methods which will force the subclass not to change the body of the method.

401
Response

Interfaces

All the fields in interfaces are public, static and final by default.
All methods are public & abstract by default.
A class that implements an interface must implement all the methods declared in the interface.
Interfaces support the functionality of multiple inheritance.

401
Response

Static Keyword

Static can be :

- Variable (also known as a class variable)
- Method (also known as a class method)
- Block
- Nested class

401
Response

Static Keyword

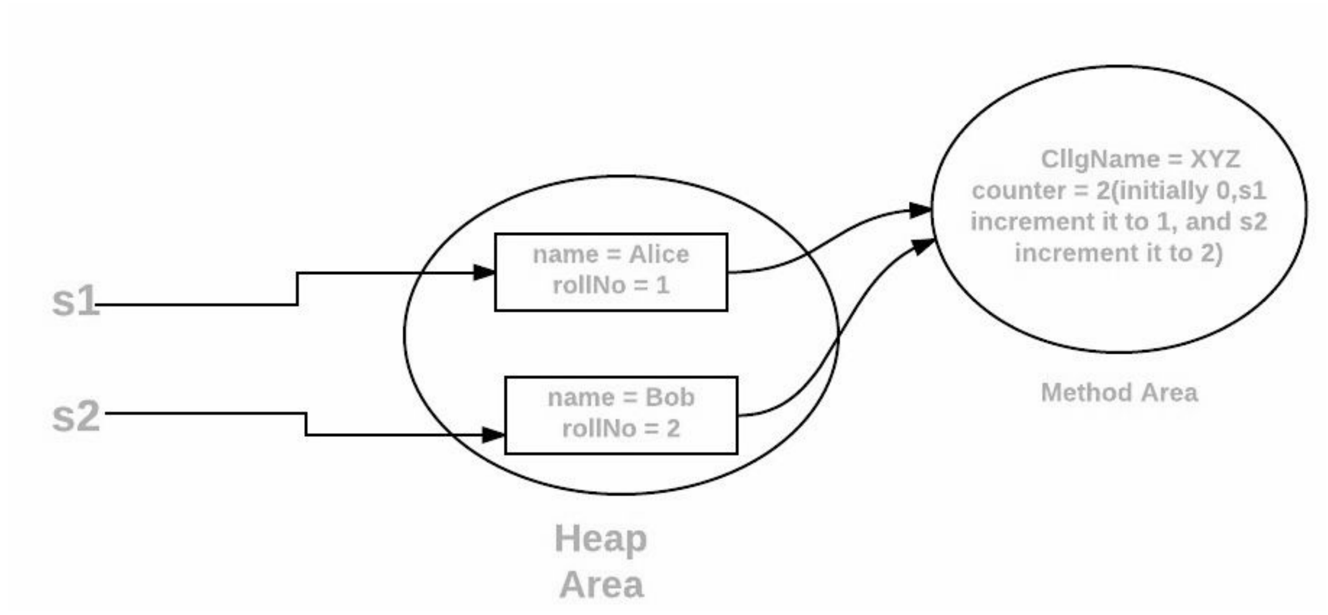
When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object

Static Block
static block that gets executed exactly once, when the class is first loaded.

401
Response

Static variables
When a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level

Static Keyword



401
Response

Static Keyword

Static methods

They can only directly call other static methods.
They can only directly access static data.
They cannot refer to this or super in any way.

401
Response

When to use static variables and methods?
Use the static variable for the property that is common to all objects

Static Classes
A class can be made static only if it is a nested class. We cannot declare a top-level class with a static modifier but can declare nested classes as static

Wrapper Classes

A Wrapper class is a class whose object wraps or contains primitive data types.

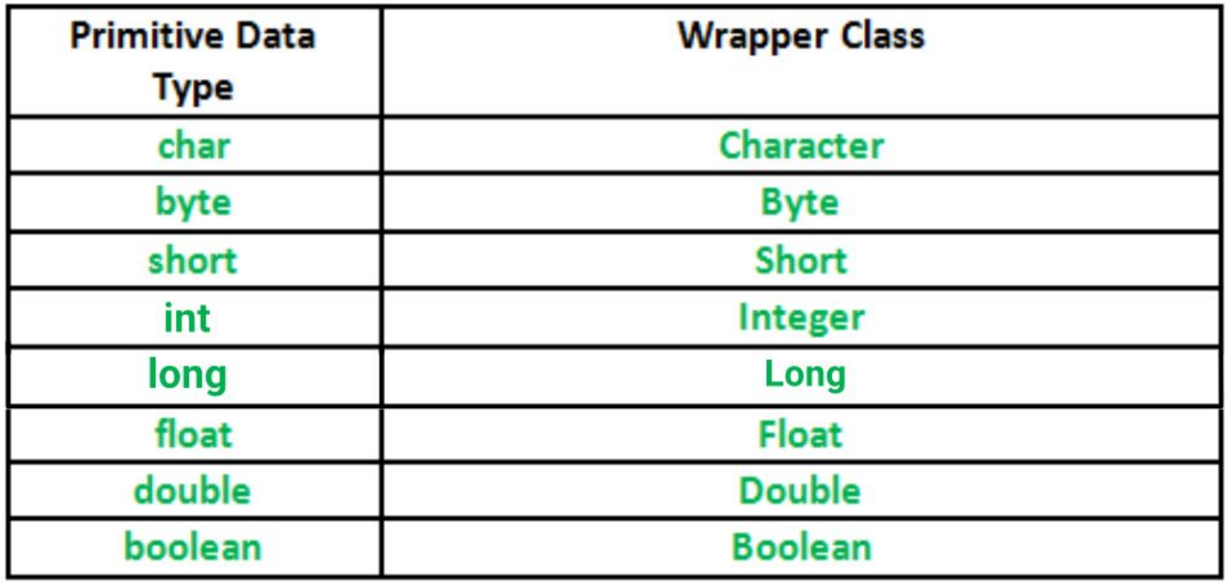
They convert primitive data types into objects.

Data structures in the **Collection framework,** such as ArrayList and Vector, store only objects (reference types) and not primitive types.

401
Response

An object is needed to support synchronization in multithreading.

Wrapper Classes



Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

401
Response

Java - Exceptions

An exception (or exceptional event) is a problem that arises during the execution of a program.

Following are some scenarios where an exception occurs.

A user has entered an invalid data.

A file that needs to be opened cannot be found.

A network connection has been lost in the middle of communications or the JVM has run out of memory.

401
Response

Checked exceptions

Unchecked exceptions

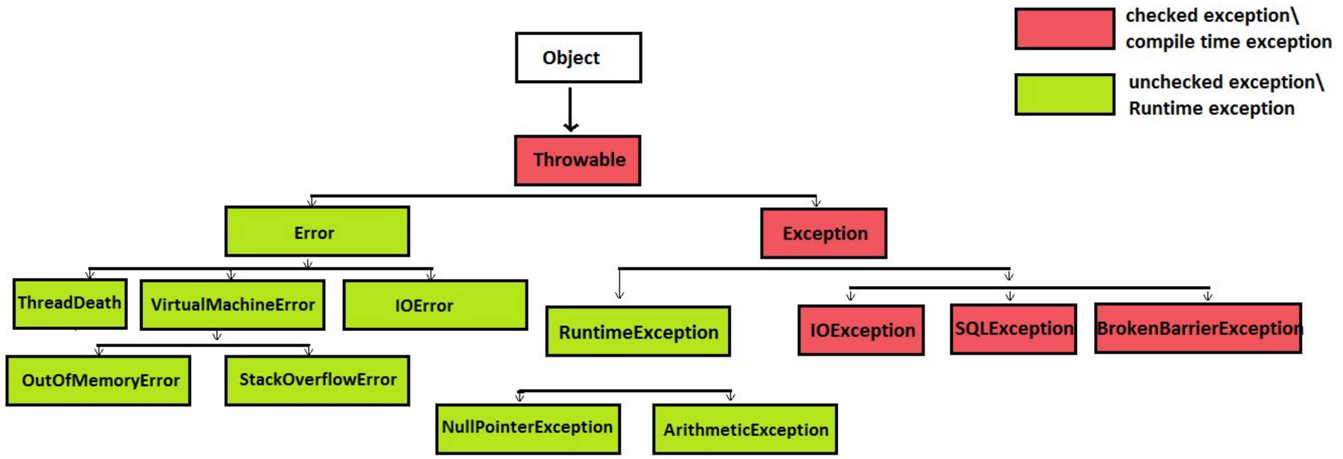
Java - Exceptions

Checked exceptions

Unchecked exceptions

(notified) by the compiler at compilation-time, these are also called as compile time exceptions.

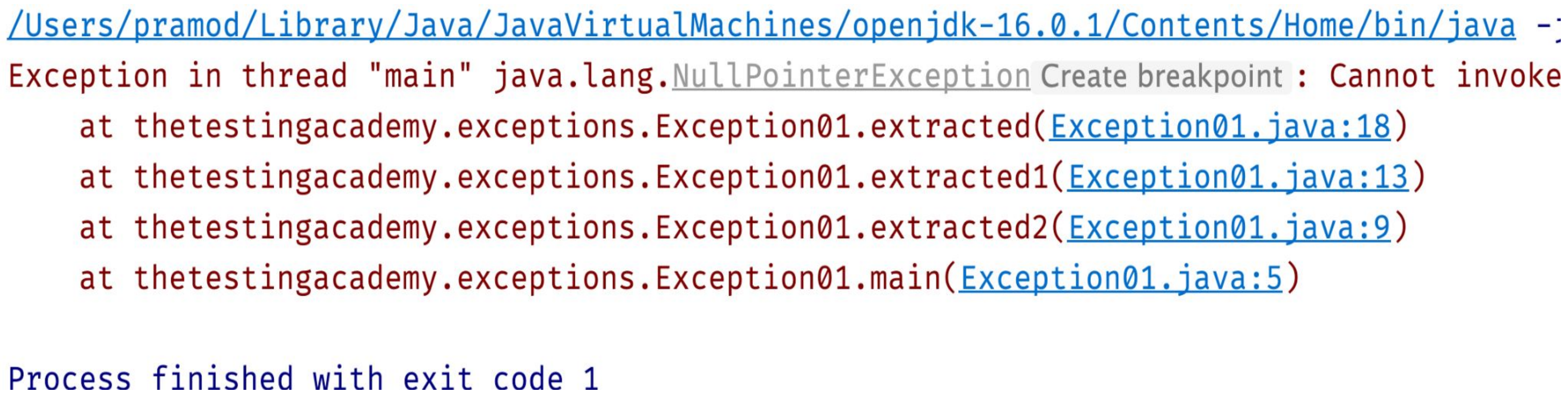
An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions



401
Response

Java - Exceptions

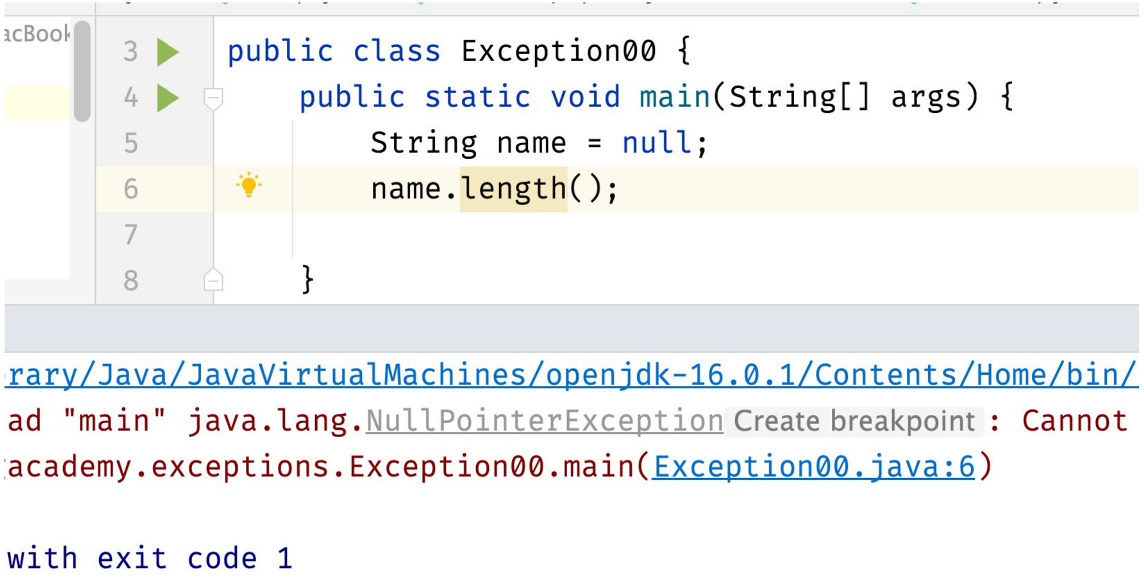
Exact line with Stack Trace



```
/Users/pramod/Library/Java/JavaVirtualMachines/openjdk-16.0.1/Contents/Home/bin/java -
Exception in thread "main" java.lang.NullPointerException Create breakpoint : Cannot invoke
	at thetestingacademy.exceptions.Exception01.extracted(Exception01.java:18)
	at thetestingacademy.exceptions.Exception01.extracted1(Exception01.java:13)
	at thetestingacademy.exceptions.Exception01.extracted2(Exception01.java:9)
	at thetestingacademy.exceptions.Exception01.main(Exception01.java:5)

Process finished with exit code 1
```

Java - Exceptions

Demo with String Null with Stack Trace



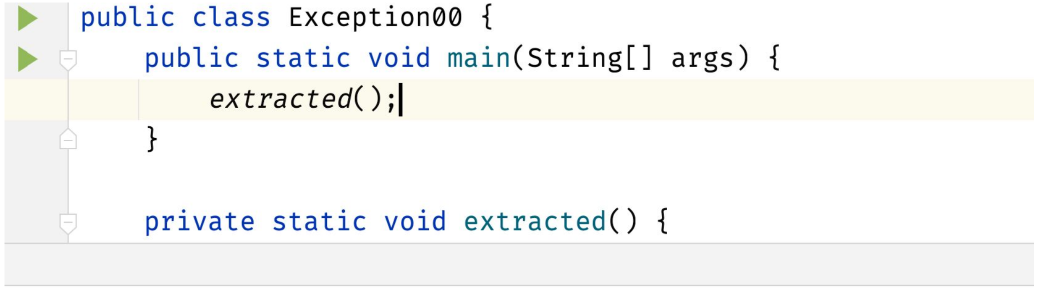
```
acBook
public class Exception00 {
    public static void main(String[] args) {
        String name = null;
        name.length();

    }
```

```
rary/Java/JavaVirtualMachines/openjdk-16.0.1/Contents/Home/bin/
ad "main" java.lang.NullPointerException Create breakpoint : Cannot
academy.exceptions.Exception00.main(Exception00.java:6)

with exit code 1
```

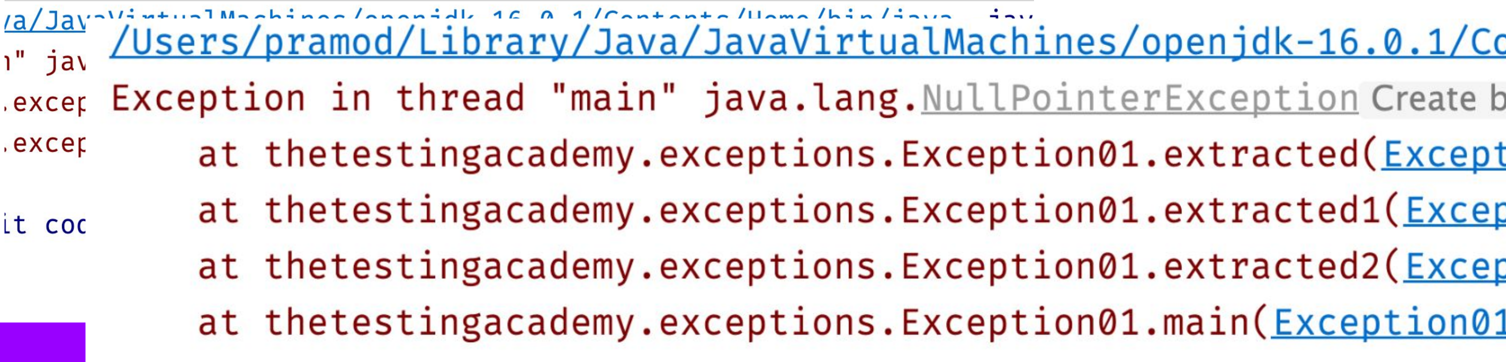
401
Response

Java - Exceptions

Exact line with Stack Trace



```
public class Exception00 {
    public static void main(String[] args) {
        extracted();
    }

    private static void extracted() {
```



```
/a/JavaVirtualMachines/openjdk-16.0.1/Contents/Home/bin/java -jav
n" jav
.excep
.excep

it coc
```

```
/Users/pramod/Library/Java/JavaVirtualMachines/openjdk-16.0.1/Co
Exception in thread "main" java.lang.NullPointerException Create b
    at thetestingacademy.exceptions.Exception01.extracted(Except
    at thetestingacademy.exceptions.Exception01.extracted1(Excep
    at thetestingacademy.exceptions.Exception01.extracted2(Excep
    at thetestingacademy.exceptions.Exception01.main(Exception01
```

401
Response

Try and Catch

Exception Handling in Java is a mechanism to handle the runtime errors so that normal flow of the application can be maintained.

It is done using 2 keywords - ‘try’ and ‘catch’.
Additional keywords like finally, throw and throws can also be used if we dive deep into this concept.

```
int[] marks = {98, 97, 95};
try {
   System.out.println(marks[4]);
} catch (Exception exception) {
   System.out.println("An exception for caught while accessing an index the 'marks' array");
}

System.out.println("We tried to print marks & an exception must have occurred with index >=3");
```

401
Response

Try and Catch

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

```
public static void main(String args[]) {
      try {
         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
```

```
try {
   // Protected code
} catch (ExceptionType1 e1) {
   // Catch block
} catch (ExceptionType2 e2) {
   // Catch block
} catch (ExceptionType3 e3) {
   // Catch block
   }
```

Catching Multiple Type of Exceptions
Since Java 7, you can handle more than one exception using a single catch block, this feature simplifies the code. Here is how you would do it −

```
catch (IOException | FileNotFoundException ex) {
   logger.log(ex);
   throw ex;
```

401
Response

The Throws/Throw & Finally Block

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

   public void withdraw(double amount) throws RemoteException,
      InsufficientFundsException {
      // Method implementation
   }
   // Remainder of class definition
}
```

Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.

Throws is used with the method signature.

401
Response

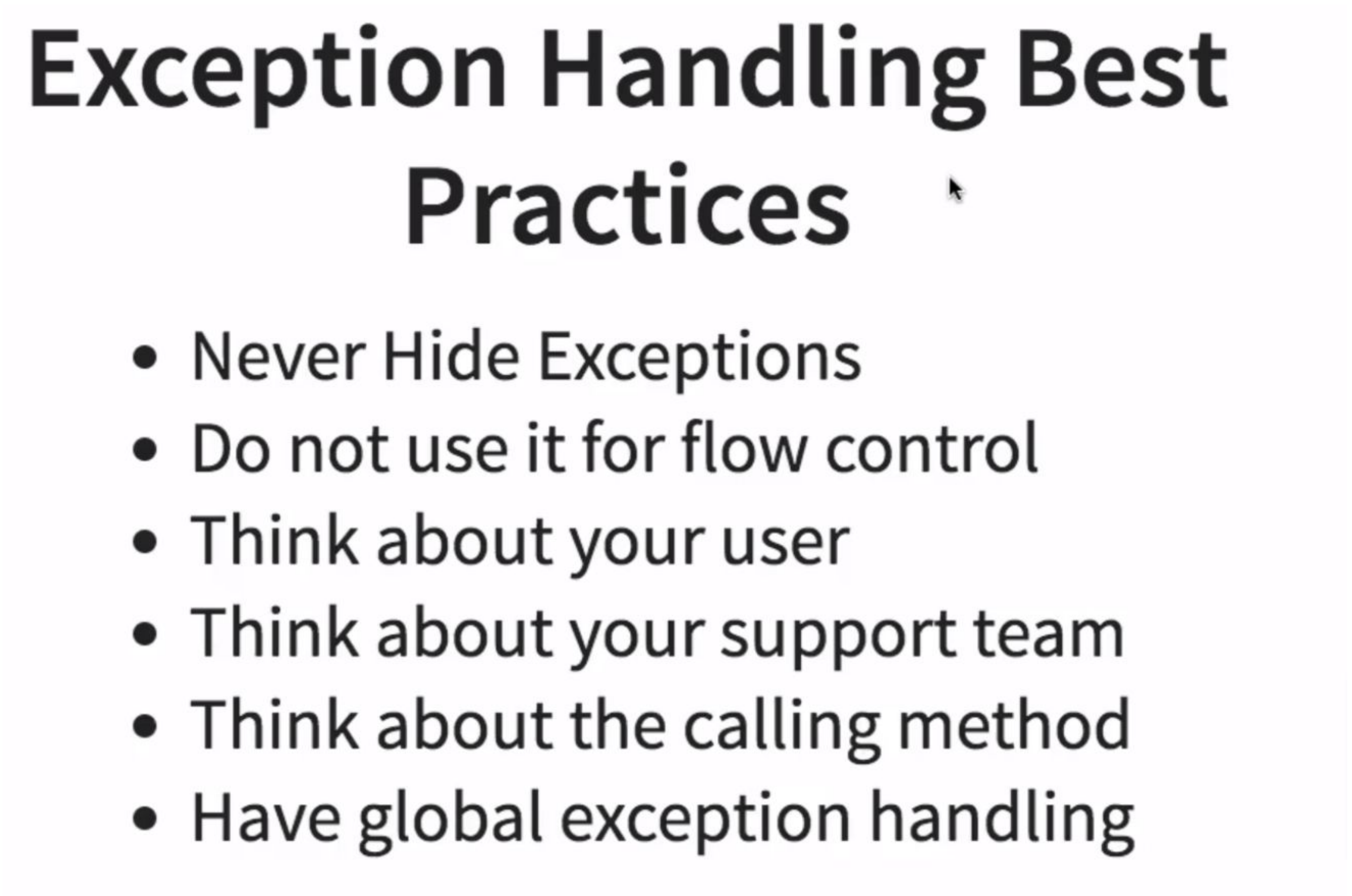
```
public void deposit(double amount) throws
RemoteException {
      // Method implementation
      throw new RemoteException();
   }
```

Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.

throw is used within the method.

Exception Hierarchy

```
//Pre-defined Java Classes
class Error extends Throwable{}
class Exception extends Throwable{}
class InterruptedException extends Exception{}
class RuntimeException extends Exception{}
class NullPointerException extends RuntimeException{}
```



Exception Handling Best Practices

- Never Hide Exceptions
- Do not use it for flow control
- Think about your user
- Think about your support team
- Think about the calling method
- Have global exception handling

401
Response

Finally Block & User-defined Exceptions

```
public static void main(String args[]) {
    int a[] = new int[2];
    try {
      System.out.println("Access element three :" + a[3]);
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Exception thrown  :" + e);
    }finally {
      a[0] = 6;
      System.out.println("First element value: " + a[0]);
      System.out.println("The finally statement is executed");
    }
  }
```

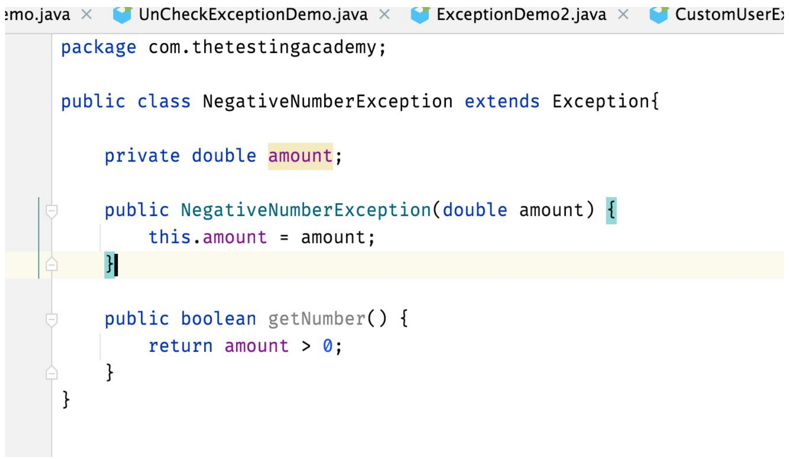
```
class MyException extends Exception {
   }
```

401
Response

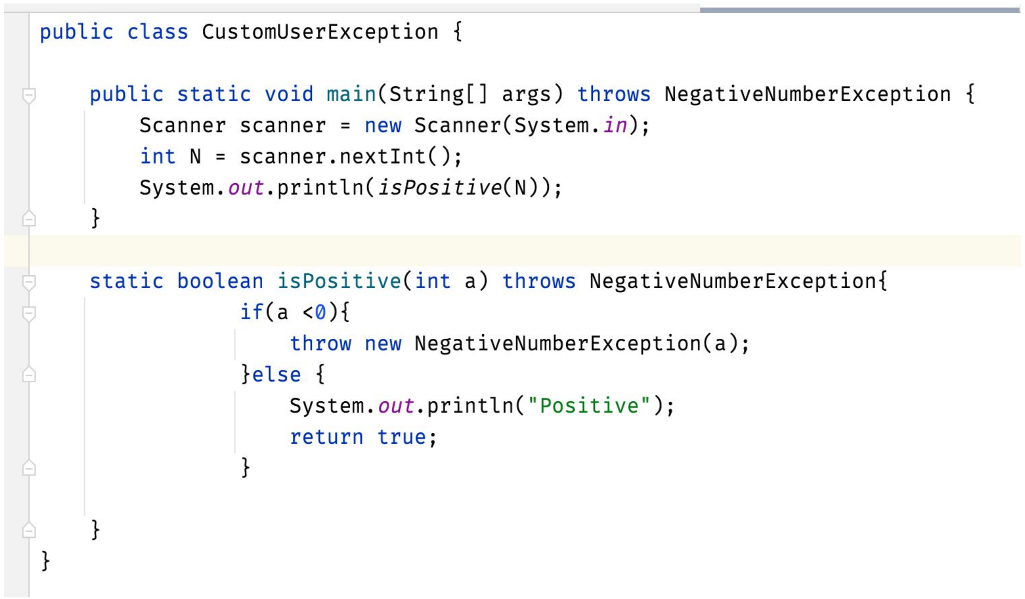
JVM Exceptions − These are exceptions/errors that are exclusively or logically thrown by the JVM. Examples: **NullPointerException, ArrayIndexOutOfBoundsException, ClassCastException.**

Programmatic Exceptions − These exceptions are thrown explicitly by the application or the API programmers. Examples: IllegalArgumentException, IllegalStateException.

Demo User-defined Exceptions



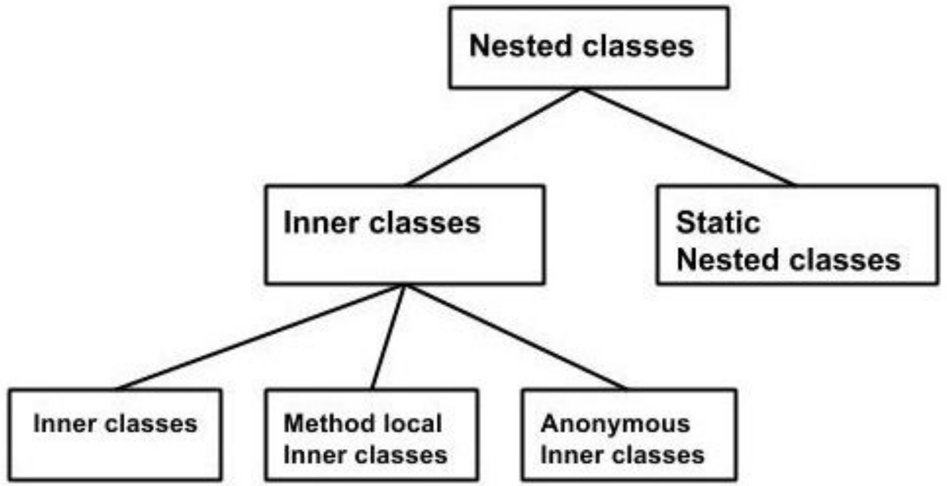
```java
package com.thetestingacademy;

public class NegativeNumberException extends Exception{

    private double amount;

    public NegativeNumberException(double amount) {
        this.amount = amount;
    }

    public boolean getNumber() {
        return amount > 0;
    }
}
```



```java
public class CustomUserException {

    public static void main(String[] args) throws NegativeNumberException {
        Scanner scanner = new Scanner(System.in);
        int N = scanner.nextInt();
        System.out.println(isPositive(N));
    }

    static boolean isPositive(int a) throws NegativeNumberException{
                if(a <0){
                    throw new NegativeNumberException(a);
                }else {
                    System.out.println("Positive");
                    return true;
                }

    }
}
```

Java - Inner classes

```
class Outer_Demo {
   class Inner_Demo {
   }
}
```



401
Response

Inner classes are a security mechanism in Java

Thanks, for attending Class

I hope you liked it. Say Thanks in Comment :)