Diagram

Description automatically generated

[JavaInterviewQuestions-UdemyCourse-September2016.pdf](file:///C:\Users\Vishnu_Kolluri\Documents\Training_ppts\java\JavaInterviewQuestions-UdemyCourse-September2016.pdf)

SmallTalk – pure OOPS

Java is not pure OOPS

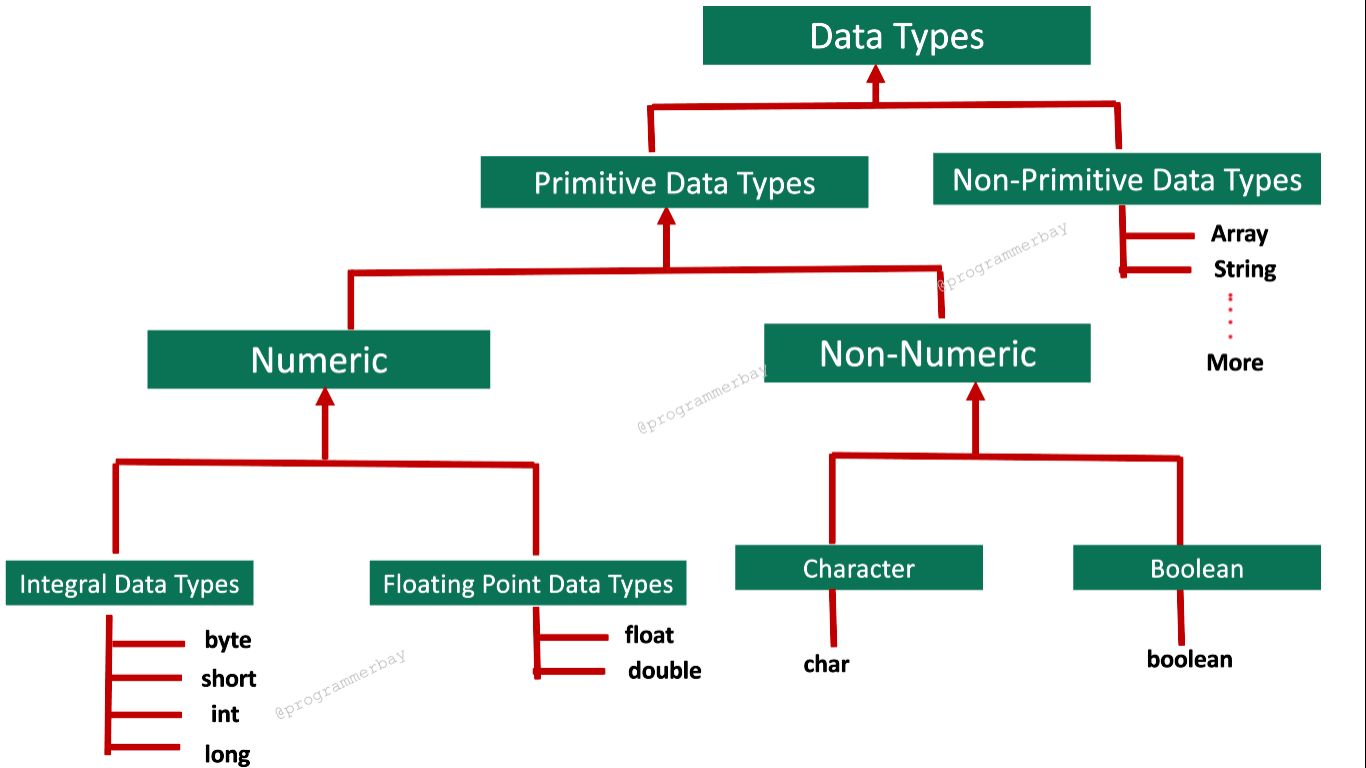
1. Due to primitive data types bcz they are not objects
2. Static Members – it breaks encapsulation
3. Control Structures (if, else, switch)
4. Non Object elements (Math, Collection)

Primitive type ----> wrapper class =Autoboxing

* Wrapper: Boolean,Byte,Character,Double,Float,Integer,Long,Short
* Primitive: boolean,byte,char ,double, float, int , long,short

Uses of Wrapper classes:

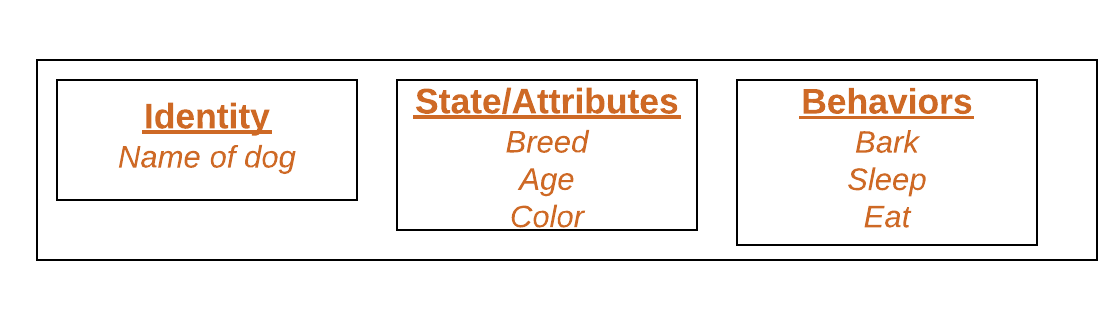
1. Can use with collections
2. Allows null values
3. Have some utility methods, can be used as objs
4. Control over memory (as they objs stored in heap memory)
5. Serialization
6. Type safe (avoid conversion errors)
7. Used with Generics



1)new 2)class.valueOf(primitivetype)

No memory is allocated when a class is declared. Memory is allocated as soon as an object is created.

OBJECT:



we can create objects in Java.

1. Using new keyword
2. Using new instance (**Class.forName)**
3. Using clone() method .clone()
4. Using deserialization readObject(b)
5. Using newInstance() method of Constructor class

EX:

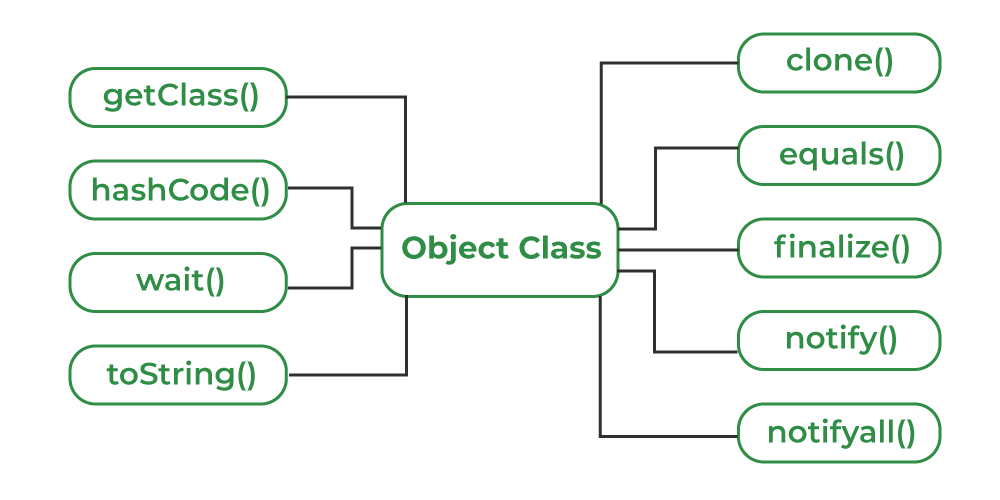
 Constructor<GFG> constructor = GFG.**class**.getDeclaredConstructor();

  GFG r = constructor.newInstance();

Java Static Block

* It is used to initialize the static data member.
* It is executed before the main() method at the time of class loading.

Object Class:

InheritanceDiagram

Description automatically generated

we can overload java main() method

**polymorphism:**

1)compile time polymorphism: Static polymorphism:Static binding: method overloading

\*(By changing number of arguments , By changing the data type)

\*Method Overloading is not possible by changing the return type

2)runtime polymorphism: dynamic polymorphism: dynamic binding: method overriding, Upcasting and Downcasting

\*The covariant return type specifies that the return type may vary in the same direction as the subclass.

parent p=new child(); or Parent p =(Parent)new Child(); (Upcasting)

Child c=(child)new parent(); (downcasting)

#### **Rules for Java Method Overriding**

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Constructors can not be overridden.

Abstract class can contain constructor but interface can not. Variables in interface cannot be private.

**Cohesion:** dependency of elements inside the class

**Coupling:** dependency of one class object on another class object. (dependent classes)

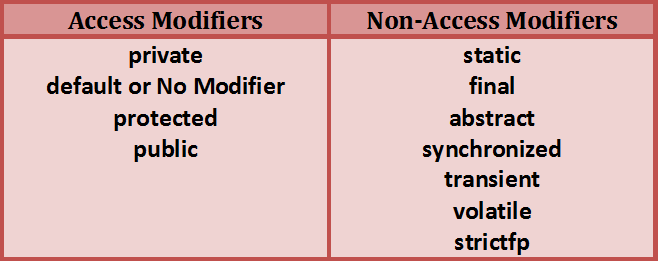
**Association:** Relation blw two separate classes through objects(one to one ,many to one etc)

**Aggregation:** weak association (weak relationship blw classes) object can exist without another object

**Composition:** Strong association(Strong relationship blw classes) object cannot exist without another object

**Constructor:**

* Constructor can be private (Singleton class) but not default keyword.
* Constructor can not be overridden.
* Can call parent class constructor from subclass constructor using super() which is automatically invoked.
* A constructor cannot be explicitly called from any method except another constructor
* A constructor should not contain return type if it contain return type it becomes method.
* A Constructor have same name as class name.
* Constructor can not be final static abstract synchronized.
* If we overload default constructor it will disappear.
* If we want to parent class constructor, it must be called in first line of constructor.



native

**synchronized** : This modifier is used to control the access of a particular method or a block by multiple threads. Only one thread can enter into a method or a block which is declared as synchronized.

**transient** : This modifier is used in serialization process. A variable which is declared as transient will not be serialized during object serialization.

**volatile** : volatile modifier is used in multi-threaded programming.

If you declare a field as volatile it will be signal to the threads that it’s value must be read from the main memory rather then their own stack.

Because volatile field is common to all threads and it will be updated frequently by multiple threads.

**strictfp** : This modifier is used for floating-point calculations. This keyword ensures that you will get same floating-point presentation on every platform. This modifier makes floating point variable more consistent across multiple platforms

**native:** The native keyword in Java is applied to a method to indicate that the method is implemented in native code using JNI (Java Native Interface)

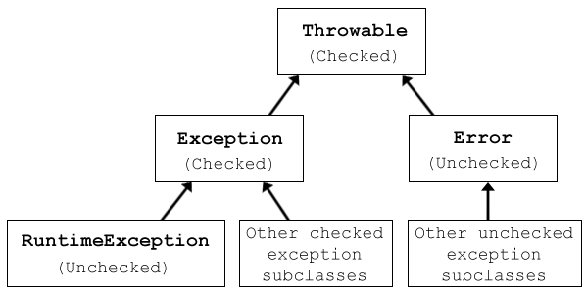
**Types of inner classes:**

1. Nested Inner Class
2. Method Local Inner Classes
3. Static Nested Classes
4. Anonymous Inner Classes

**Exception Handling:** Runtime exceptions are unchecked exceptions

Diagram

Description automatically generated



**Custom Exception :**

public class AgeException extends Exception{

public AgeException(String str){

super(str);

}

}

Then throw new AgeException(“Under Age”);

* For Static Default value is used. byte=0,int=0,float=0.0,double=0.0,boolean=false, long=0,char=null

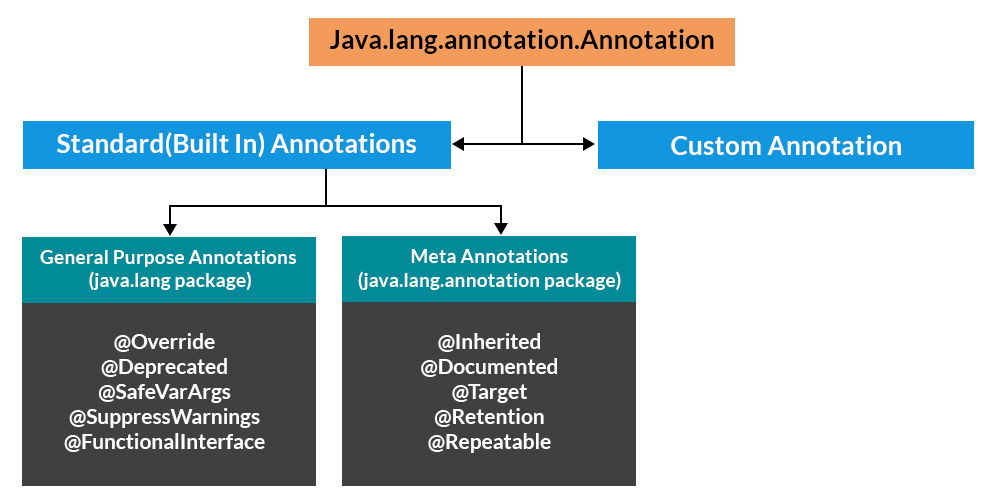
If SuperClass doesn’t declare any exception and SubClass can declare **Unchecked exception but not checked excep.**

* The subclass can throw the same or smaller exceptions.
* It can choose not to throw any exceptions.
* It cannot throw new checked exceptions not in the parent method.

**Annotations:**

There are broadly 5 categories of annotations as listed:

1. Marker Annotations
2. Single value Annotations
3. Full Annotations
4. Type Annotations
5. Repeating Annotations



Implementing user-defined annotations.

1. **Annotation Name**is an interface and Annotation methods can’t have parameters
2. The parameter should not be associated with method declarations and **throws**clause should not be used with method declaration.
3. Parameters will not have a null value but can have a default value.
4. ***default value***is optional.
5. The return type of method should be either primitive, enum, string, class name, or array of primitive, enum, string, or class name type.

We will use Reflection to parse java annotations from a class. Please note that Annotation Retention Policy should be RUNTIME

**Generics:**

<https://www.digitalocean.com/community/tutorials/java-generics-example-method-class-interface>

We can’t write code as List<Number> numbers = new ArrayList<Integer>();

it will cause errors in achieving Type safety.

Using <? extends Number> we can do it.

For subtyping always use Upper or lower bound and wildcards

### we can’t create generic array or write code as List<Integer>[] array = new ArrayList<Integer>[10];

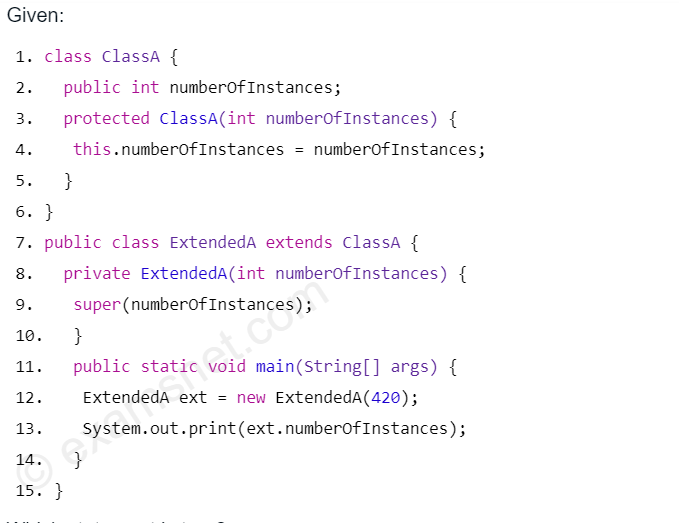
This information is used at runtime to throw ‘ArrayStoreException’ if elements type doesn't match to the defined type. Since generics type information gets erased at compile time by Type Erasure, the array store check would have been passed where it should have failed.

* Generics in Java was added to provide type-checking at compile time and it has no use at run time,
* Java compiler uses **type erasure** feature to remove all the generics type checking code in byte code and insert typecasting if necessary.
* Type erasure ensures that no new classes are created for parameterized types; consequently, generics incur no runtime overhead.



interface Abc{  
 public abstract void play();  
  
}  
abstract class player implements Abc{  
 public abstract void play();  
}  
class Team extends player{  
  
 @Override  
 public void play() {  
 System.*out*.println("Main");  
 }  
}  
public class Main {  
 public static void main(String[] args) {  
 Abc t=new Team();  
 t.play(); //output is Main.

//Abstract class can implement interface  
 }  
   
 }

****

//o/p :420

* We can access variables from subclass access modifier best example
* Can’t find length() of null, it gives Null Pointer Exception.
* We can declare the**inner classes** as **private** or **protected**, but it is not allowed in **outer** **classes**.

Graphical user interface, text, application, email

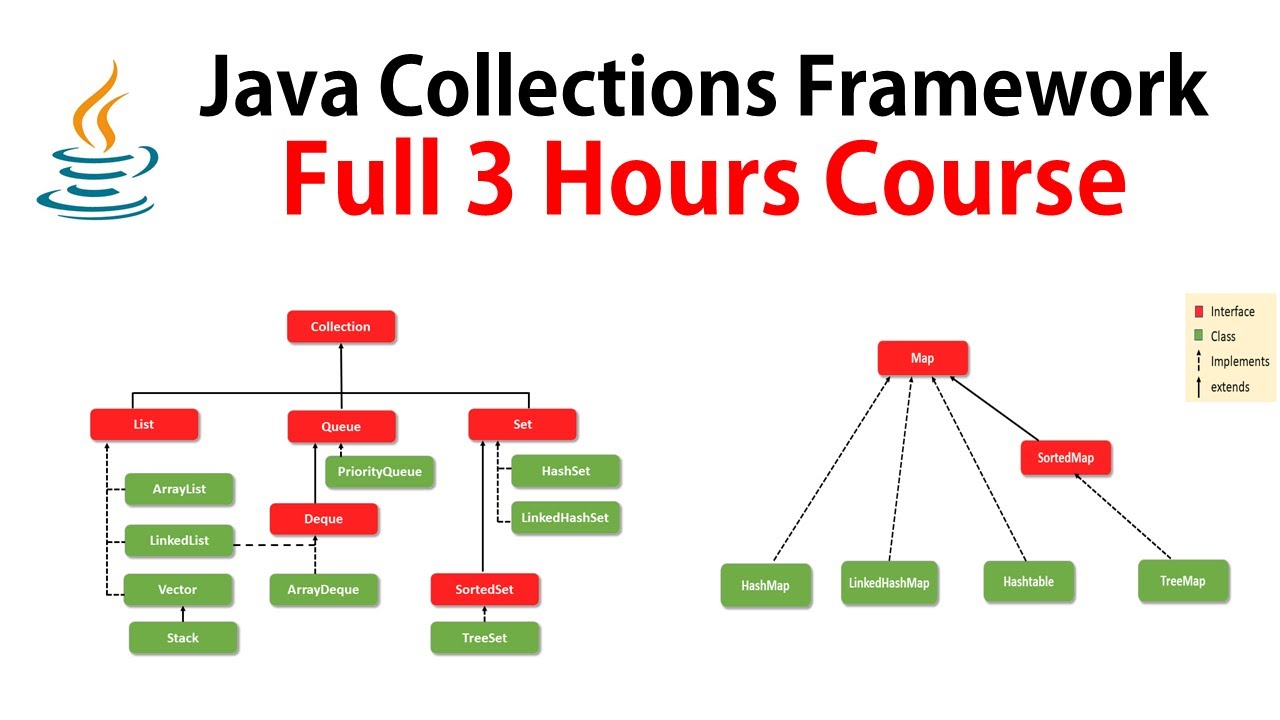
Description automatically generated

Prints parent first and then children.

class One {  
 public One() {  
 System.*out*.print(1);  
 }  
 public void play() {  
 System.*out*.println("play1");  
 }  
  
}  
class Two extends One {  
 public Two() {  
 System.*out*.print(2);  
 }  
 public void play(){  
 System.*out*.println("play2");  
 }  
}  
class Three extends Two {  
 public Three() {  
 System.*out*.print(3);  
 }  
 public void play(){  
 System.*out*.println("play3");  
 }  
}  
  
public class Main {  
  
 public static void main(String[] args) {  
 new Three().play();  
 }  
  
}

olp: 123play3

**Collections:**



Lambdas vs anonymous:

<https://www.geeksforgeeks.org/difference-between-anonymous-inner-class-and-lambda-expression/>

so return type of lambdas is function right

**Uses of lambdas:**

* Lambdas are used to avoid boiler plate code
* Access to functional programming
* Function statements
* Memory is reduced bcz anonymous classes creates $.class files but lambdas doesn’t create .class files.

"boilerplate code" is any seemingly repetitive code that shows up again and again in order to get some result or reusable code. Like method signatures

**Design Principles:**

S – single responsibility principle

O – Open closed principle

L - Liskov substitution principle

I – Interface Segregation principle

D – Dependency inversion principle

KISS – Keep It Simple Stupid.

YAGNI – You Aren’t Gonna Need It.

DRY – Don’t Repeat Yourself. (Use constants and methods)

WET – Write Every Time. (Instead of importing code from another project we should write again)

<https://www.digitalocean.com/community/tutorials/gangs-of-four-gof-design-patterns>

Text

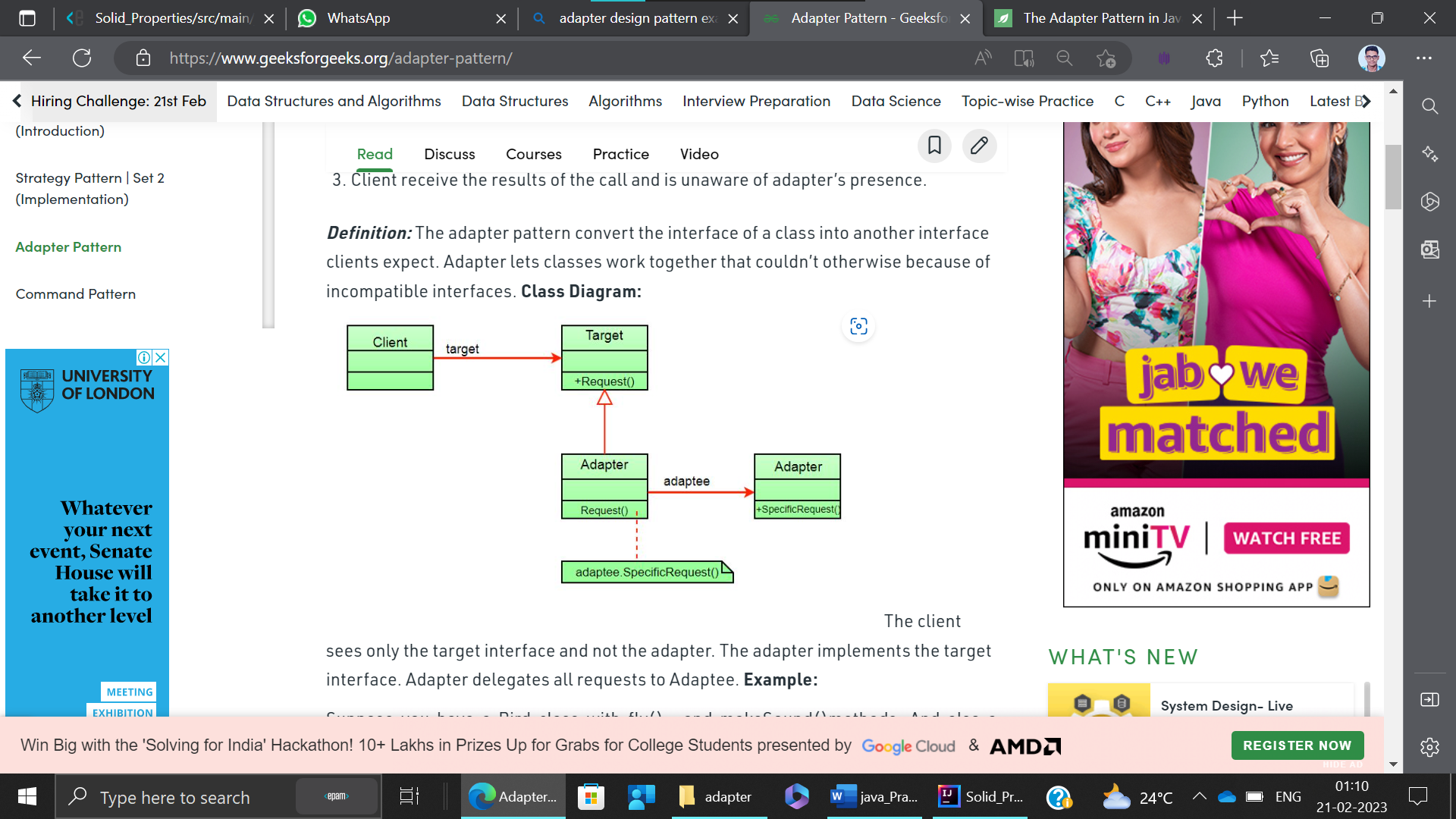
Description automatically generated

Text

Description automatically generated with medium confidence

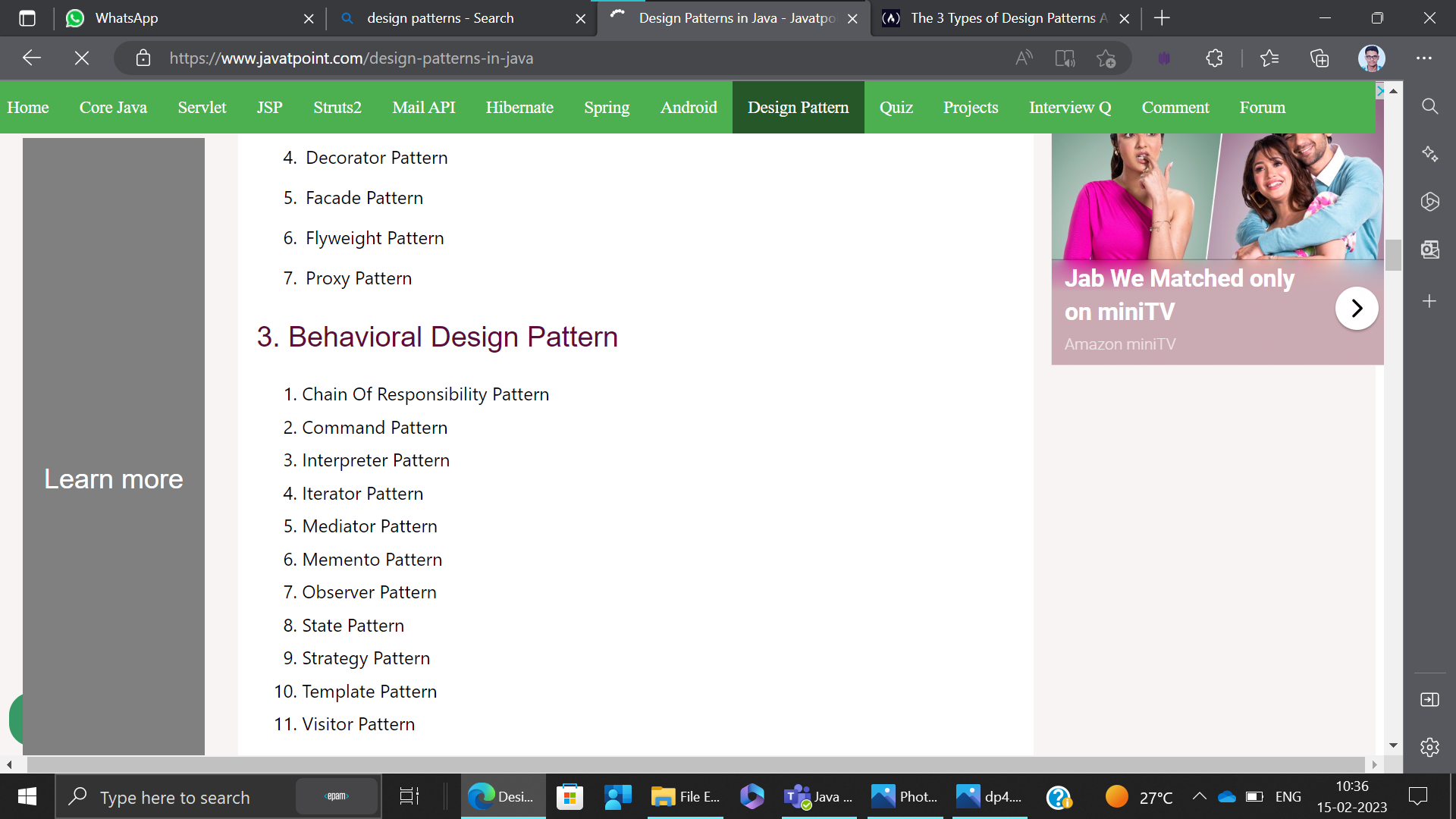
Graphical user interface, text

Description automatically generated



Graphical user interface, text, application

Description automatically generated



Text

Description automatically generated

In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides published a book titled Design Patterns - Elements of Reusable Object-Oriented Software which initiated the concept of Design Pattern in Software development. These authors are collectively known as Gang of Four (GOF).

**Event handling frameworks like swing, awt use Observer Pattern**

Facade pattern structural hides the complexities of the system and provides an interface to the client using which the client can access the system.

Manual Testing:

<https://youtube.com/playlist?list=PLUDwpEzHYYLseflPNg0bUKfLmAbO2JnE9>

**JDBC:**

**DriveManager:**

It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.

**To store Image:**

For storing image into the database, BLOB (Binary Large Object) datatype is used in the table

CREATE TABLE  "IMGTABLE"

   (    "NAME" VARCHAR2(4000),

    "PHOTO" BLOB

   ) ;

You can store and retrieve images in the database in java by the help of **PreparedStatement** interface.

The **setBinaryStream()** method of PreparedStatement is used to set Binary information into the parameterIndex.

**public** **void** setBinaryStream(**int** paramIndex,InputStream stream,**long** length)

**throws** SQLException

Retrieve image: javatpoint

The **getBlob()** method of PreparedStatement is used to get Binary information, it returns the instance of Blob.

After calling the **getBytes()** method on the blob object, we can get the array of binary information that can be written into the image file.

**public** Blob getBlob()**throws** SQLException

**public**  **byte**[] getBytes(**long** pos, **int** length)**throws** SQLException

file

CLOB(character large object)

ps.setCharacterStream(2,fr,(**int**)f.length());

getClob()

**Rest assured.io:**

[Home · rest-assured/rest-assured Wiki · GitHub](https://github.com/rest-assured/rest-assured/wiki/)

http://hamcrest.org/JavaHamcrest/javadoc/1.3/org/hamcrest/Matchers.html

Graphical user interface, text, application, email

Description automatically generated

A picture containing whiteboard

Description automatically generated

**TestNG:**

@Test(enabled=false)

Public void testMethod(){

sout(“in testmethod”);}

****