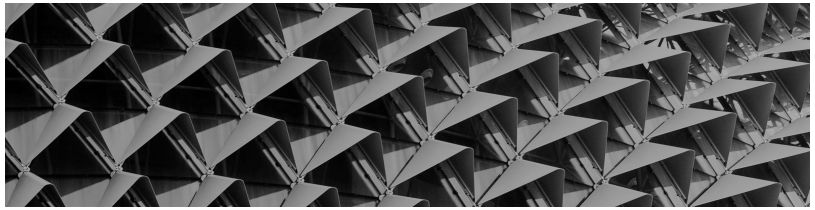
OOPS with Java

Before we begin



- What is a class ?
- What is an object ?
- What is Abstraction ?
- What is Encapsulation ?
- Inheritance Vs Composition ?
- Inheritance
- What is Polymorphism ?
- Polymorphism
- Interface

- Basic programming knowledge

What is OOPS

A programming paradigm based upon objects (having both data and methods together) that aims to incorporate the *advantages of modularity and reusability*. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

What is a class & object?

What is a class?

A **class** is a template or blueprint to build a specific type of object.

What is a object?

An **object** is an instance of a class.

A class is a template for objects.

Characteristics of a Class

Properties: Variables declared in the class

Methods: Functions used to expose the behavior.

Constructors: A special type of method to create *instances* of the objects.

Example of a Class

The Car Class

What are some **properties** of a car?

What are some **behaviours** of a car?

Example of a Class

The Car Class

What are some **properties** of a car?

- color
- price
- numberOfDoors

What are some **behaviours** of a car?

- start()
- stop()
- addGas(liters)

Parts of an object

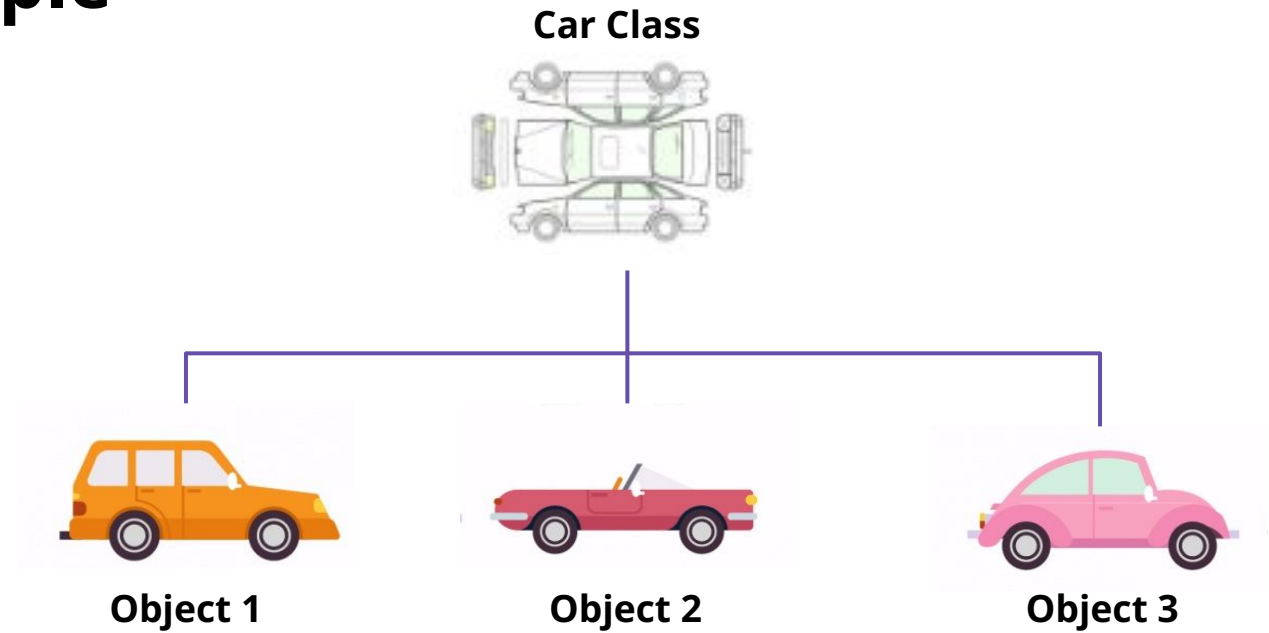
```java
public class Car {
    private String style;
    private String color;
    private int numberOfDoors;
    private int speed;

    public Car() {
    }

    public Car(String style, String color, int numberOfDoors) {
        this.style = style;
        this.color = color;
        this.numberOfDoors = numberOfDoors;
    }

    public void accelerateSpeed(int speed) {
        this.speed += 5;
    }

    public void brakeSpeed(int speed) {
        this.speed -= 5;
    }

    public int getSpeed() {
        return this.speed;
    }
}
```

State

Constructor

Behaviour

Example



Notice that from a single Car **class**, we can create multiple *different* car **objects**. Remember: a class is just a *blueprint* for creating objects. All objects of a given class have the same properties, but not necessarily the same values for those properties (state of each object can be different)!

Break Out Activity

Create a class named Student with following properties:

- Name
- Roll Number
- Marks

Create following behaviours for the class:

- checkIfStudentPassedOrFailed

Create Application class and within main method using objects of Student class, set the value of student name, roll number and marks. Then with the help of student class behavior ‘checkIfStudentPassedOrFailed’ print based on marks of a student if he failed or passed.

Input : Name, Roll Number and Marks

Output: Passed or Failed

Constructor

DEFAULT CONSTRUCTOR

A constructor with no arguments is called a default constructor. It is the constructor that is defined implicitly by the compiler. When used all the object's properties are initialized with the default value for its type.

PARAMETERIZED CONSTRUCTOR

This type of constructor is used when we want to initialize the object's properties with certain values. These values can be passed to the constructors as parameters. These constructs are also helpful in *constructor overloading**.

Difference between constructor and method

CONSTRUCTOR	METHOD
Is used to initialize the state of an object	Is used to expose the behavior of an object
Must not have a return type	Must have a return type
Is invoked implicitly	Is invoked explicitly
The name must be same as the class name	The name may or may not be same as class name

Creating Object

<Class> <variableName> = new <Class>(<constructor arguments>);

Ex:-

```
Class Car {
        String color;

        public Car() { }  // If a parameterized constructor is defined, default constructor has to defined explicitly

        public Car(String color) {
                this.color = color;
        }
}


Car car = new Car();   // Using default constructor.

Car blueCar = new Car("blue");  // Using parameterized constructor
```

Naming conventions

Class names should be in mixed case with the first letter of each internal word capitalized.

Object reference (variables) names should be in mixed case with a lowercase first letter

https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

Car.java

```
class Car {

  String color;

  public Car(){ }

  public Car(String color){
    this.color = color;
  }

  public String getColor() {
    return this.color;
  }
}
```

Application.java

```
class Application {

  public static void main(String[] args) {
    Car car = new Car("blue");
    System.out.println(car.getColor());

    Car yetToBePainted = new Car();
    System.out.println(yetToBePainted.getColor());
  }

}
```

```
$ javac Application.java Car.java
$ java Application
```

Break Out Activity

1. Create a Student class with following properties
    a. Name
    b. Roll Number
    c. Marks
2. Add default and parameterized constructor
3. Create Application class with main method
4. Create few objects of Student in Application class
5. Print state of Student objects

Static

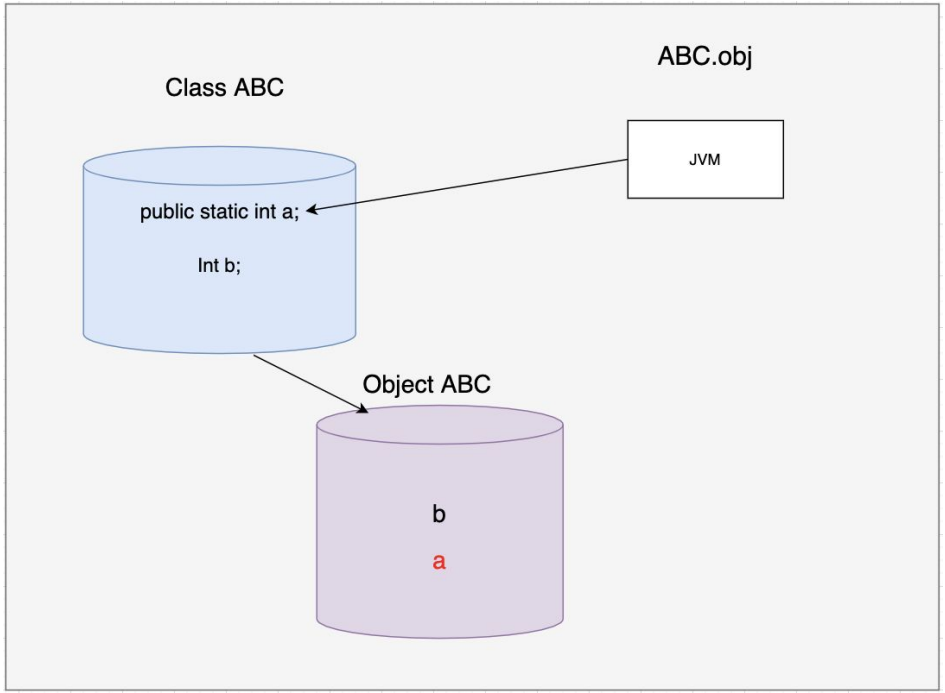
The `static` keyword

Static simply means "not part of an object", or "not dynamically created" as part of an object when the object is created.

Static Fields

When a field is static, its value is shared by all objects of a class. Sometimes it's referred to as "a class variable".

There is only *one* instance of this field across all instances



Static Fields

Static fields behave differently from the typical instance fields.

STATIC FIELDS	INSTANCE FIELDS
Shared by all objects of the class	Part of each object instance
`Class.variableName`	`objectVariable.variableName`
One value only	May have different values for each object
Stored in only one place	Stored in the memory of each object
"Statically" allocated before any objects are created	"Dynamically" allocated each time an object is created

Static Methods

- In Java, a static method is **a method that belongs to a class rather than an instance of a class**. ... A static method is not part of the objects it creates but is part of a class definition. Unlike instance methods, a static method is referenced by the class name and can be invoked without creating an object of class.

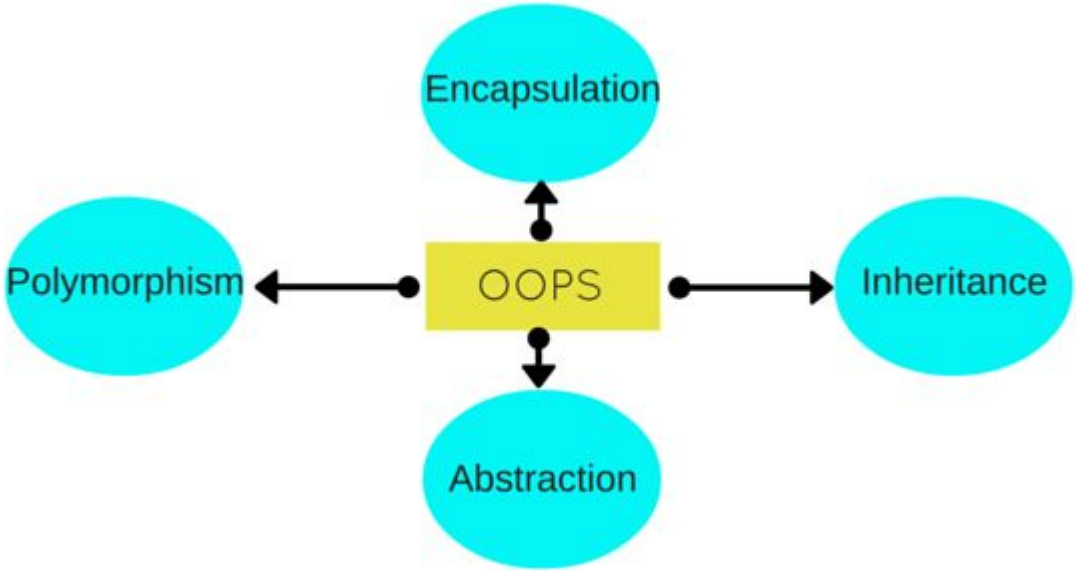
- The main method that starts a Java program is always static. It usually creates the first objects. There is only one main method for each execution of a Java program.

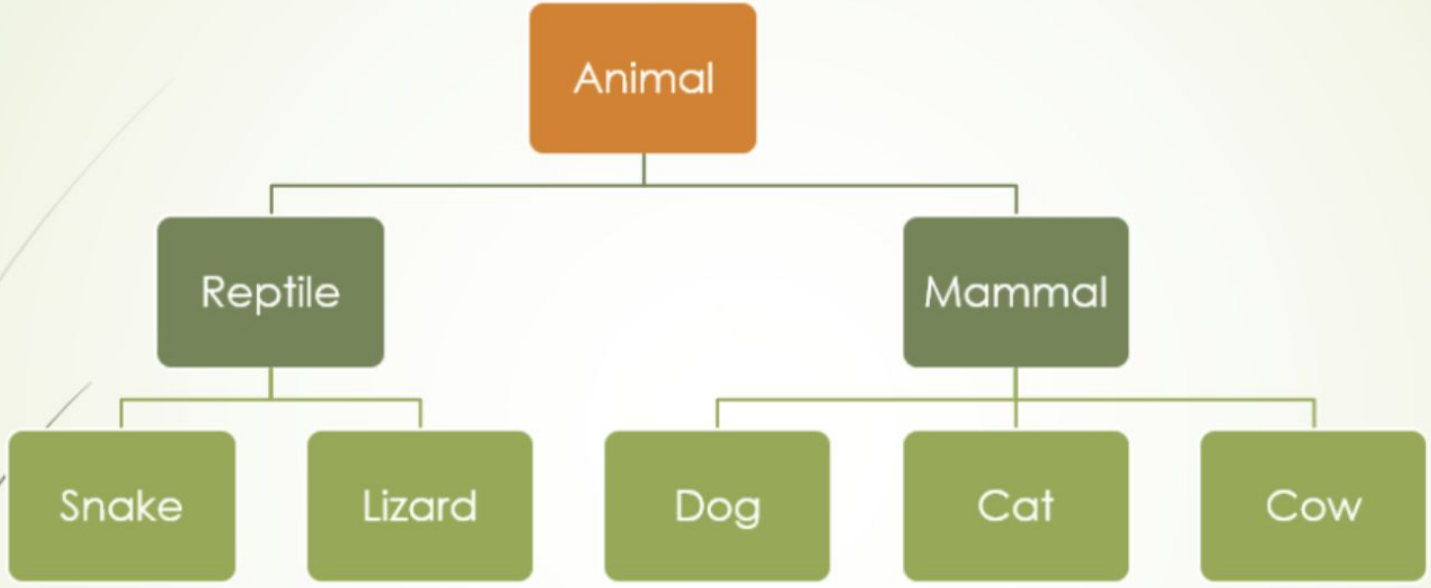
Break Out Activity

Add numberOfStudents static parameter to Student class.

Increment it when a new Student is created

Print number of students created so far in application class

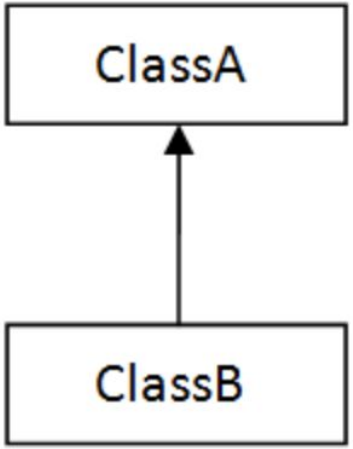

Encapsulation
Polymorphism
OOPS
Inheritance
Abstraction



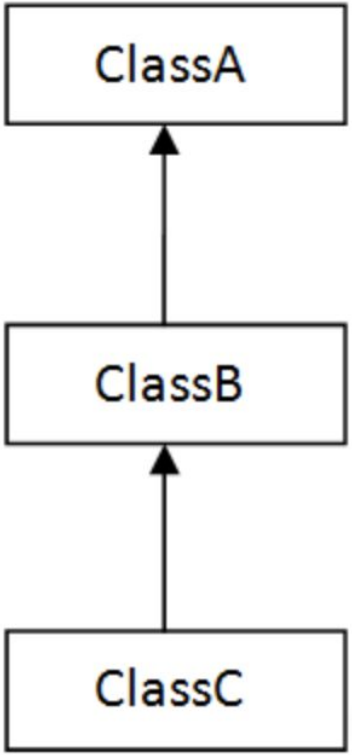
Inheritance

INHERITANCE

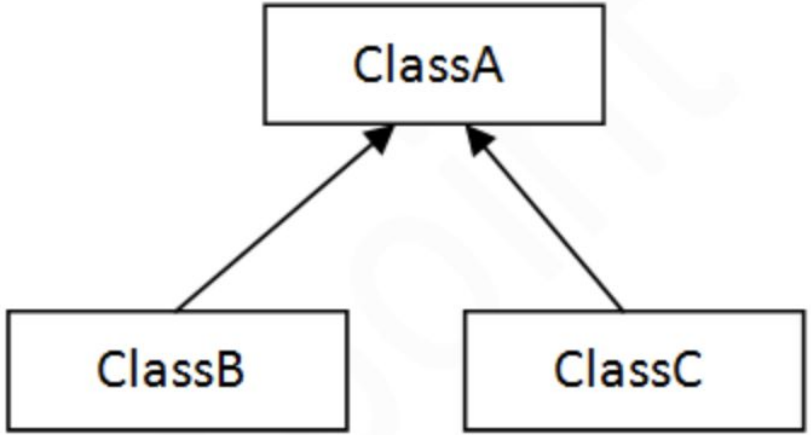
Inheritance is one of the core concepts of object-oriented programming (OOP) languages. It is a **mechanism where you can to derive a class from another class for a hierarchy of classes that share a set of attributes and methods**.



1) Single



2) Multilevel



3) Hierarchical

Why Inheritance?

The idea of inheritance was as a way of code reuse.

Dog inherited code from Animal. Poodle inherited code from Dog. That way, code that was common to all Animals lives in Animal, all Dogs lives in Dog, and stuff special to Poodles lives only in Poodle.

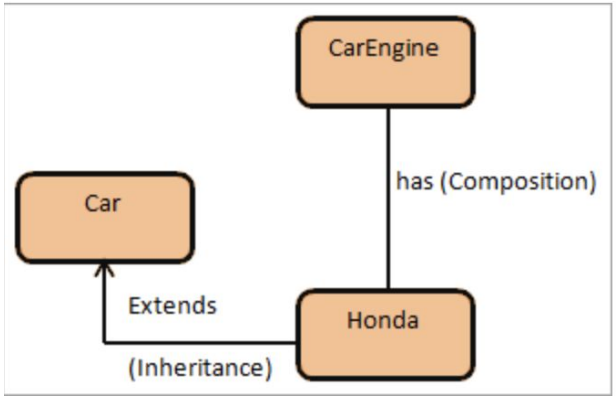
Inheritance vs Composition?

In object-oriented programming, we can use inheritance when we know there is an "is a" relationship between a child and its parent class. Some examples would be:

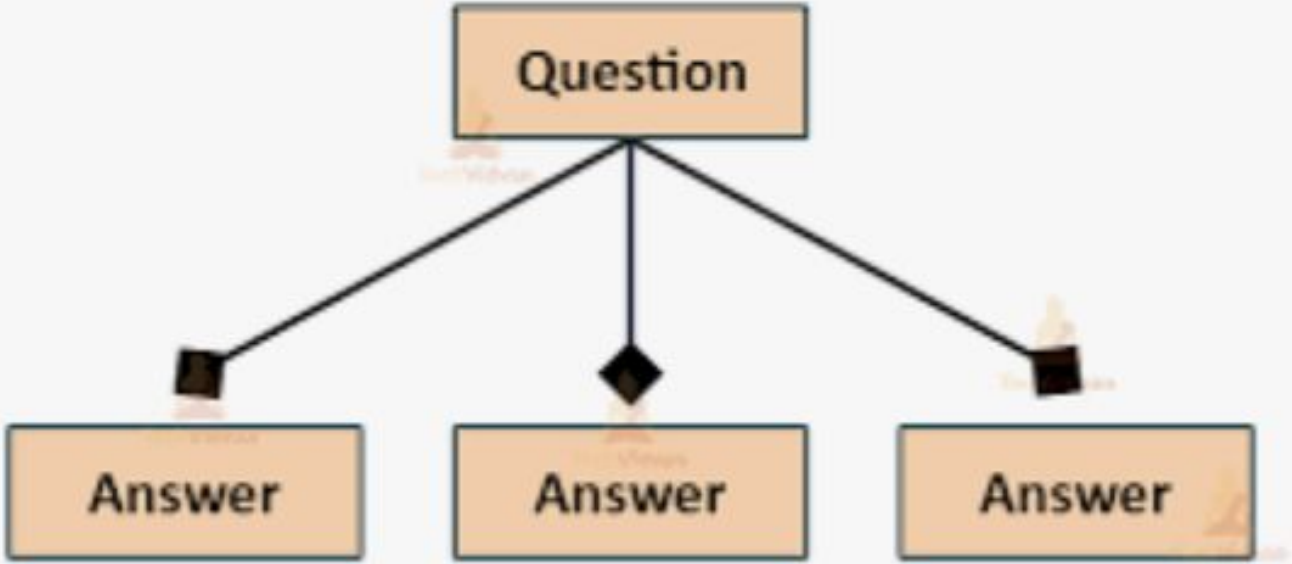
- A person *is a* human.
- A cat *is an* animal.
- A car *is a* vehicle.

In object-oriented programming, we can use composition in cases where one object "has" (or is part of) another object. Some examples would be:

- A car *has a* battery (a battery *is part of* a car).
- A person *has a* heart (a heart *is part of* a person).
- A house *has a* living room (a living room *is part of* a house).



Composition in Java



Break Out Activity

From Parent class student, extend 2 child class called ScienceStudents and CommerceStudents.

Remove marks properties from Parent class and add physics, chemistry and maths marks properties to ScienceStudents class and add accounts, economics and businessStudies marks properties to CommerceStudents class.

Create objects for ScienceStudents class and CommerceStudents class to populate respective subject marks and then print them.

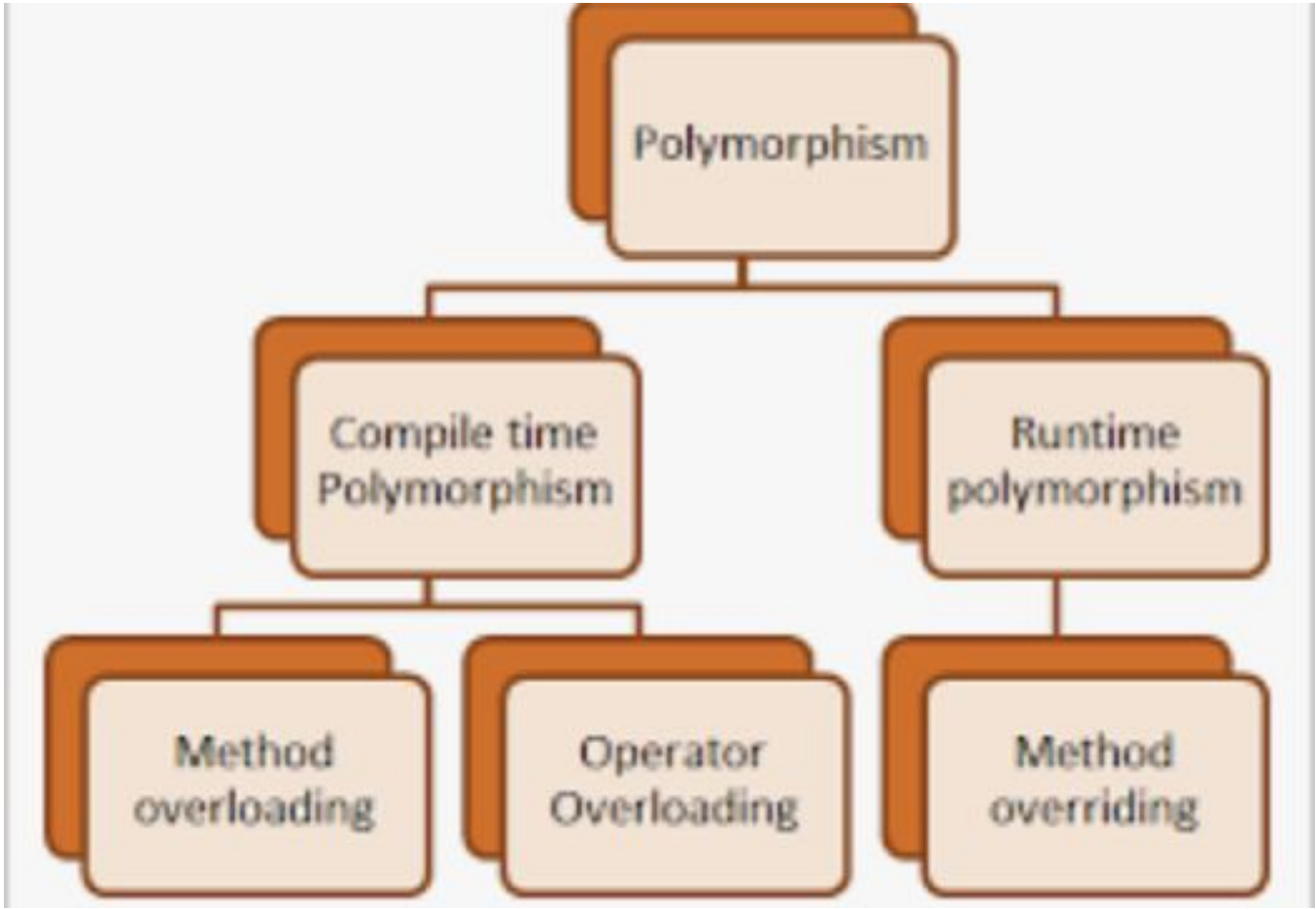
ACCESS MODIFIERS

Access modifier specifies the scope/accessibility of a variable or a method or a class from the same class or from a different class from a same or different package

Level	Private	Default	Protected	Public
Same Class	Y	Y	Y	Y
Same Package Extended	N	Y	Y	Y
Same Package Non-Extended	N	Y	Y	Y
Diff Package Extended	N	N	Y	Y
Diff Package Non-Extended	N	N	N	Y

Polymorphism

Polymorphism is **the ability of any data to be processed in more than one form**. The word itself indicates the meaning as poly means many and morphism means types.


Polymorphism
Compile time Polymorphism
Runtime polymorphism
Method overloading
Operator Overloading
Method overriding

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.

Break Out Activity

Create 2 methods with same name “addition” where:

1) First addition method should sum 2 numbers
2) Second addition method should concatenate 2 strings

Call these methods from an application class with main method

Encapsulation

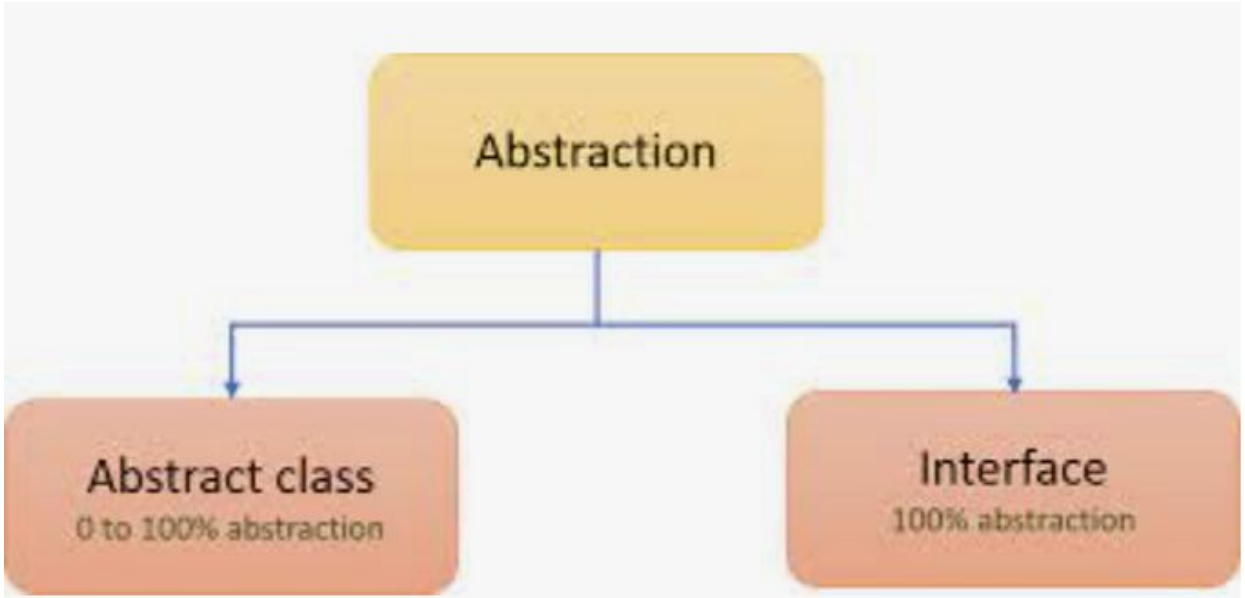
In object-oriented programming, encapsulation refers to the bundling of data with the methods that operate on that data, or the restricting of direct access to some of an object's components.

Break Out Activity

For previously created CommerceStudent and ScienceStudent child classes, create getter and setter methods to get and set respective subject marks.

Abstraction

In Java, Data Abstraction is defined as **the process of reducing the object to its essence so that only the necessary characteristics are exposed to the users**.



Abstract class

An abstract class is **a class that is declared abstract** —it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

Interface

The interface in Java is **a mechanism to achieve abstraction**. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables.

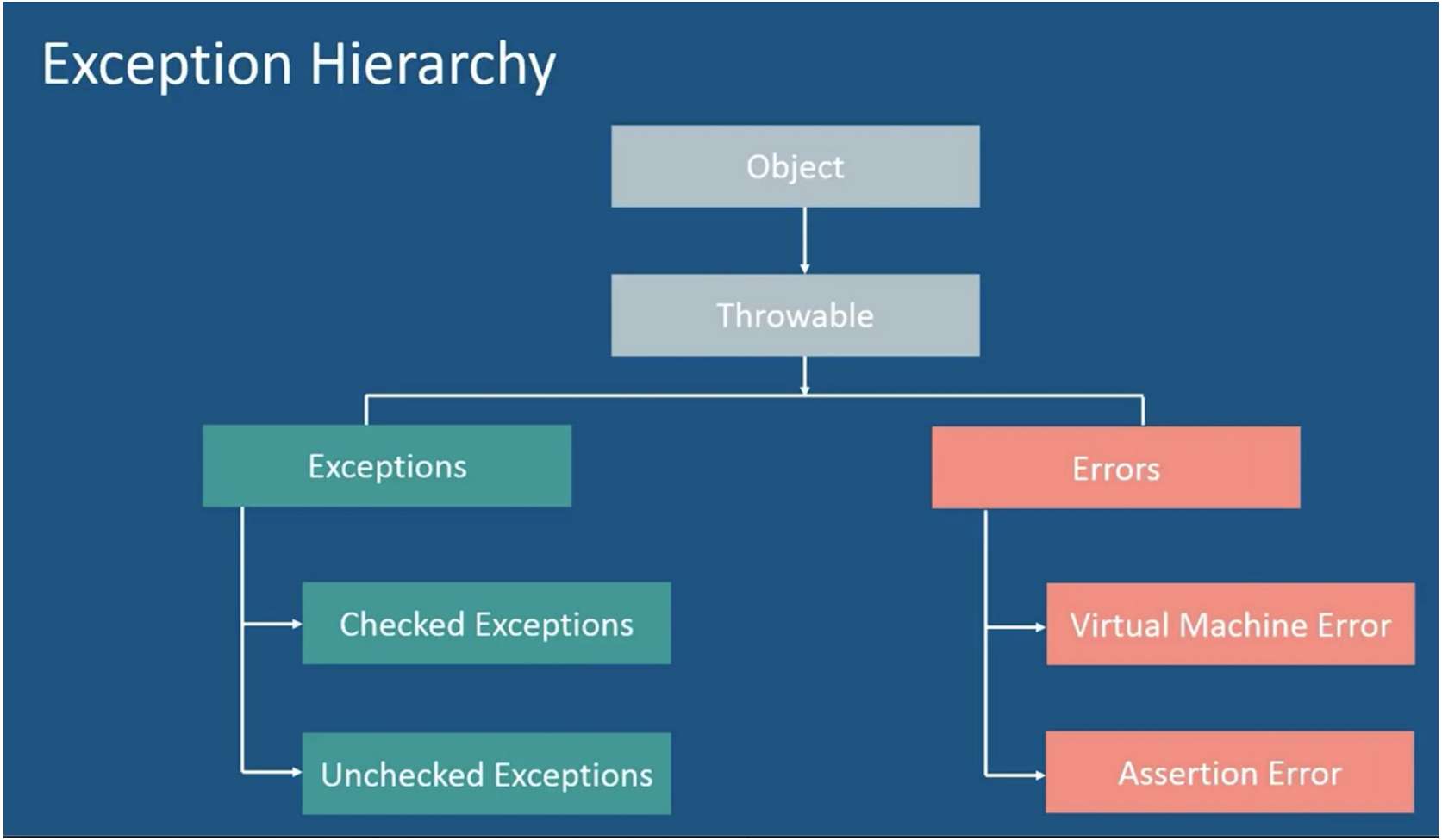
Break Out Activity

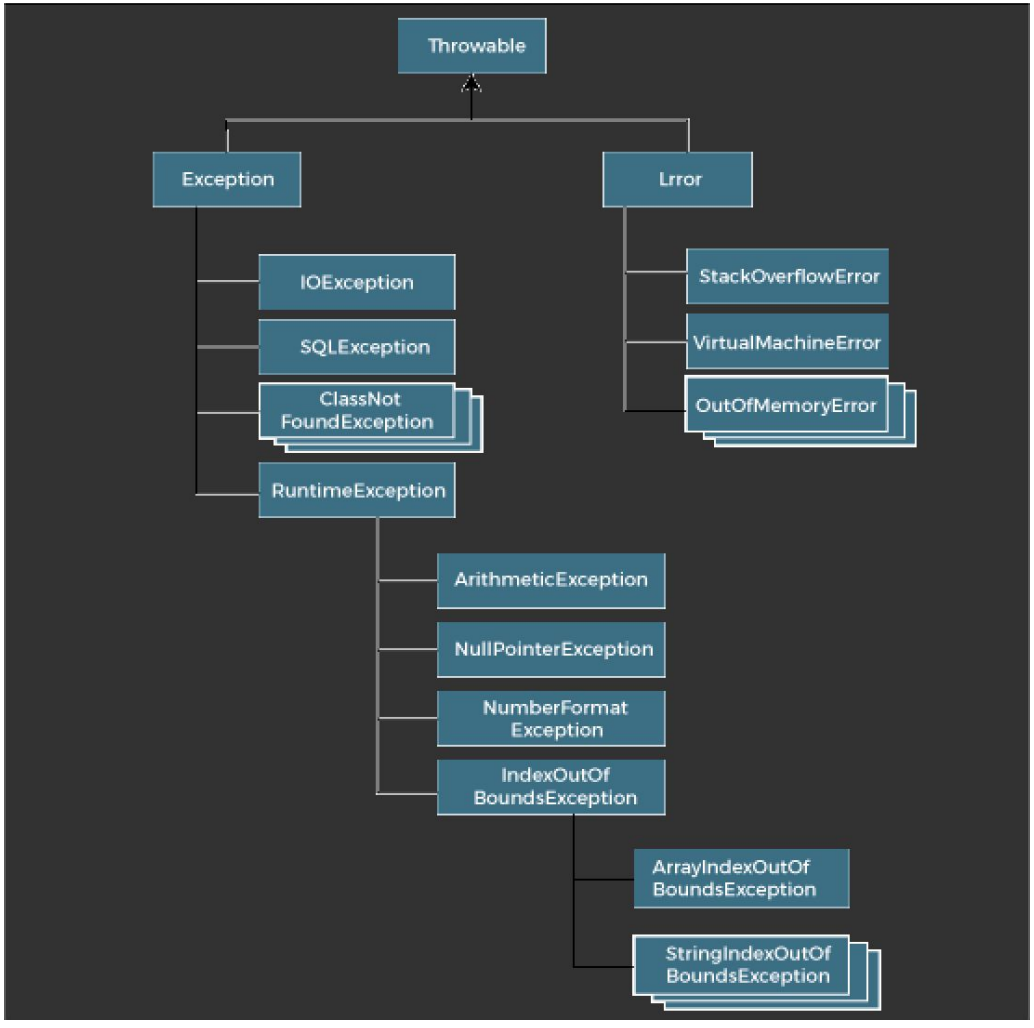
Create an Animal Interface and implement 2 classes called Dog and Cat.

Exception Handling

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

Exception Hierarchy




Throwable
Exception
Lrror
IOException
SQLException
ClassNot FoundException
RuntimeException
StackOverflowError
VirtualMachineError
OutOfMemoryError
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException

Why Exception handling?

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

statement 9;

statement 10;

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed.