# Review of OOP in Java

# What is an Object?

- An entity that encapsulates state and behavior
  - **State (data):** variables inside the object
    - **field**: A variable inside an object that is part of its state
      - Each object has its own copy of each field.
  - **Behavior (methods):** methods inside the object
    - **instance method**: Exists inside each object of a class and gives behavior to each object.

## Objects & References

- “Reference semantics”
  - When one variable is assigned to another, the **object is not copied**; both variables **refer to the SAME *object***
  - Modifying the value of one variable **will** affect others
  - Benefits: efficiency and sharing
- For example, pass around an array as a parameter

## What Defines an Object?

- Class: A program entity that represents either
  - A program / module (e.g., a client program), or
  - A template/blueprint for a new type of objects

# Implicit Parameter – this

- The object on which an instance method is being called
- Refers to the implicit parameter inside the class
  - Refer to a field: `this.field`;
  - Call a method: `this.method(parameters);`
  - One constructor can call another: `this(parameters);`

## Types of Instance Methods

- **Accessor/Getter**: A method that allow clients to examine object state
- **Mutator/Setter**: A method that modifies an object's state
- **Helper**: Assists some other method in performing its task
  - often declared as private so outside clients cannot call it

## Constructor

- Initializes the state of new objects
- No return type
- Triggered by the `new` keyword in the client program
- A class can have multiple constructors
  - Overloading constructors

    Each one must accept a unique set of parameters

## Inheritance

- Forming new classes based on existing ones.
  - A way to share/**reuse code** between two or more classes

- **Subclass extends Superclass**
  - **Superclass**: Base class being extended
  - **Subclass**: Derived class that inherits all fields and methods from superclass
  - **is-a relationship**: Each object of the subclass also "is a(n)" object of the superclass and can be treated as one (i.e., polymorphism).

## Inherited Private Fields and Methods

- Inherited private fields/methods **cannot be directly accessed** by subclasses
  - Solution 1: Set accessor modifier to **protected** in the superclass
  - Solution 2: Implemented **getter methods** in the superclass

## Keyword – super

- A subclass can call its superclass' method/constructor (and hence the superclass version of these methods)
  - Call a method: `super.method(parameters);`
  - Call a constructor: `super(parameters);`

# Polymorphism

- A class can implement an **inherited** method in **its own way**
- Allows a variable of a **superclass** type to **refer to** an object of one of its **subclasses**, and determines **which overridden method to execute** depending on data types
  - `<SuperclassType> <objName> = new <SubclassName>();`

## Overriding Methods

- To write a new version of a method in a subclass that replaces the superclass's version.
  - Method name, return type, parameters must match exactly

- `public String toString() { … }`
- `public boolean equals(Object obj) { … }`

## Overriding toString()

- Returns the String representation of an object
- The default version in Object Class
  - <ClassName>@<MemoryAddress>
- Overridden version in subclasses
  - Depending on the program description

# Overriding equals(Object obj)

- The default version in Object Class
  - Compares the referential equality, just like ==
- Overridden version in subclasses
  - Keyword `instanceof` → if the parameter is an instance of the target type
  - Object casting → if yes, treat it as the target type
  - Compare some or all fields depending on the program description

Q: Which point is FALSE from the following?

[✓ A.] A class is an object
B. A class is a template or prototype that defines the composition and the behavior of all objects of certain kinds
C. A class may have fields of composite types
D. From a class you may initiate an object

Q: Select ALL of the constructors that would be valid for a class named Puppy:

A. public MyPuppy (String name) {…}
[✓ B. public Puppy (String name, int age) {…}]
C. public void Puppy (String name) {…}
[✓ D. public Puppy (String name) {…}]
[✓ E. public Puppy (String name, Color color) {…}]
F. public int Puppy (String name, int age) {…}

Q: Why should we use encapsulation? (Select ALL that apply)

- [✓ A.] Can later change the internal workings of the class without modifying client code
- B. Clients can directly access the fields
- [✓ C.] Clients cannot directly access or modify its internal workings – nor do they need to do so
- [✓ D.] Protects data from unwanted access
- [✓ E.] Encapsulation leads to abstraction

```java
public class Car {
    public void m1() {
        System.out.println("Car 1");
    }

    public void m2() {
        System.out.println("Car 2");
    }

    public String toString() {
        return "It's a Car";
    }
}

public class Truck extends Car {
    public void m1() {
        System.out.println("Truck 1");
    }
}
```

## Q: What's the output of the client code?

```java
Car mycar = new Car();
Truck mytruck = new Truck();

System.out.println(mycar);   // It's a Car
mycar.m1();   // Car 1
mycar.m2();   // Car 2

System.out.println(mytruck); // It's a Car
mytruck.m1();   // Truck 1
mytruck.m2();   // Car 2
```

## Given the Movie Class →→→

Q: Write a VideoLibrary class that represents a list of Movie objects. No javadoc required.

Assume the list will have no more than 5 Movies. DO NOT use magic numbers, create and use a class constant.

The VideoLibrary class should have the following private fields:

- Movie[] movieList – an array of Movies.
- **int** numMovies – number of Movies currently in the movieList.

```
public class Movie {
    private String title;
    private int length; //in minutes

    public Movie(String title, int length) {
        this.title = title;
        this.length = length;
    }

    public int getLength() {
        return length;
    }
}
```

```
public class VideoLibrary {

    //TODO: Add code...

    public VideoLibrary() {
        //TODO: Add code...
    }

    public void add(Movie movie) {
        //TODO: Add code...
    }

    public int getTotalLength() {
        //TODO: Add code...
    }
}
```

## Given the Movie Class →→→

Q: Write a VideoLibrary class that represents a list of Movie objects. No javadoc required.

Assume the list will have no more than 5 Movies. DO NOT use magic numbers, create and use a class constant.

The VideoLibrary class should have the following private fields:

- Movie[] movieList – an array of Movies.
- **int** numMovies – number of Movies currently in the movieList.

```java
public class Movie {
    private String title;
    private int length; //in minutes

    public Movie(String title, int length) {
        this.title = title;
        this.length = length;
    }

    public int getLength() {
        return length;
    }
}
```

```java
public class VideoLibrary {

    //TODO: Add code...

    public VideoLibrary() {
        //TODO: Add code...
    }

    public void add(Movie movie) {
        //TODO: Add code...
    }

    public int getTotalLength() {
        //TODO: Add code...
    }
}
```

## Given the Movie Class →→→

Q: Write a VideoLibrary class that represents a list of Movie objects. No javadoc required.

Assume the list will have no more than 5 Movies. DO NOT use magic numbers, create and use a class constant.

The VideoLibrary class should have the following private fields:

- Movie[] movieList – an array of Movies.
- **int** numMovies – number of Movies currently in the movieList.

```
public class Movie {
    private String title;
    private int length; //in minutes

    public Movie(String title, int length) {
        this.title = title;
        this.length = length;
    }

    public int getLength() {
        return length;
    }
}

public class VideoLibrary {

    public static final int MAX_MOVIES = 5;
    private Movie[] movieList;
    private int numMovies;

    public VideoLibrary() {
        //TODO: Add code...
    }

    public void add(Movie movie) {
        //TODO: Add code...
    }

    public int getTotalLength() {
        //TODO: Add code...
    }
}
```

## Given the Movie Class →→→

Q: Write a VideoLibrary class that represents a list of Movie objects. No javadoc required.

The VideoLibrary class should have the following methods:

- **public** VideoLibrary() - Constructs a new empty VideoLibrary. Initializes numMovies to 0.
- **public void** add(Movie movie) – Adds the given Movie to the VideoLibrary's movieList if the list has fewer than 5 items, otherwise it does nothing.
- **public int** getTotalLength() – returns the total length of all Movies currently in the VideoLibrary (using the Movie class accessor method).

```
public class Movie {
    private String title;
    private int length; //in minutes

    public Movie(String title, int length) {
        this.title = title;
        this.length = length;
    }

    public int getLength() {
        return length;
    }
}

public class VideoLibrary {

    public static final int MAX_MOVIES = 5;
    private Movie[] movieList;
    private int numMovies;

    public VideoLibrary() {
        //TODO: Add code...
    }

    public void add(Movie movie) {
        //TODO: Add code...
    }

    public int getTotalLength() {
        //TODO: Add code...
    }
}
```

## Sample Solution

```java
public class Movie {
    private String title;
    private int length; //in minutes

    public Movie(String title, int length) {
        this.title = title;
        this.length = length;
    }

    public int getLength() {
        return length;
    }
}
```

```java
public class VideoLibrary {

    public static final int MAX_MOVIES = 5;
    private Movie[] movieList;
    private int numMovies;

    public VideoLibrary() {
        movieList = new Movie[MAX_MOVIES];
        numMovies = 0;
    }

    public void add(Movie movie) {
        if (numMovies < MAX_MOVIES) {
            movieList[numMovies] = movie;
            numMovies++;
        }
    }

    public int getTotalLength() {
        int length = 0;
        for (int i = 0; i < numMovies; i++) {
            length += movieList[i].getLength();
        }
        return length;
    }
}
```