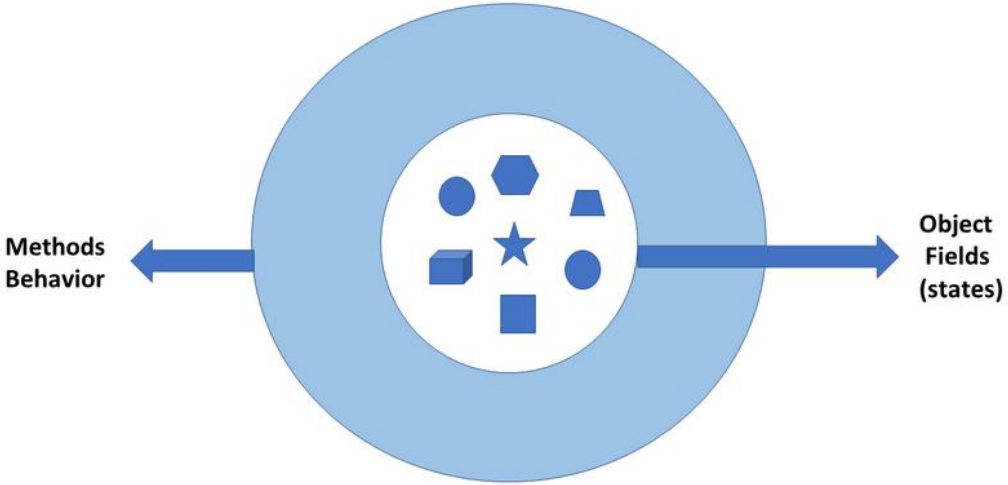
OOP concepts

Object-Oriented programming (OOP)

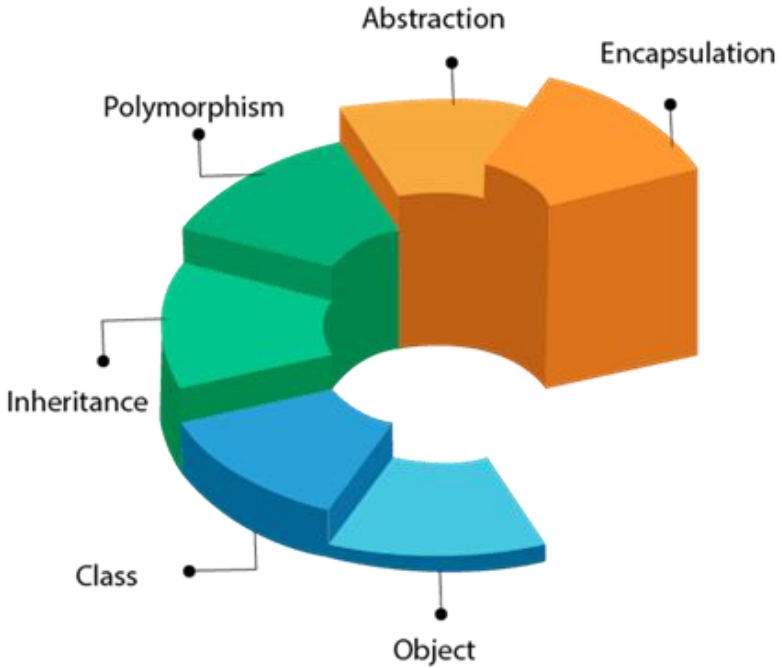
Object-oriented programming (OOP) is a programming methodology that is based on the concept of **objects**, which are components that possess an **identity**, a **state** and a **behaviour**.



Object-oriented programming (OOP) involves programming using **objects**.

OOP concepts

OOP simplifies software development and maintenance by providing some concepts:



OOP concepts

Quick Reference

Abstraction is the process of exposing the essential details of an entity, while ignoring the irrelevant details, to reduce the complexity for the users.

Encapsulation is the process of bundling data and operations on the data together in an entity.

Inheritance derives a new type from an existing class, thereby establishing a parent-child relationship.

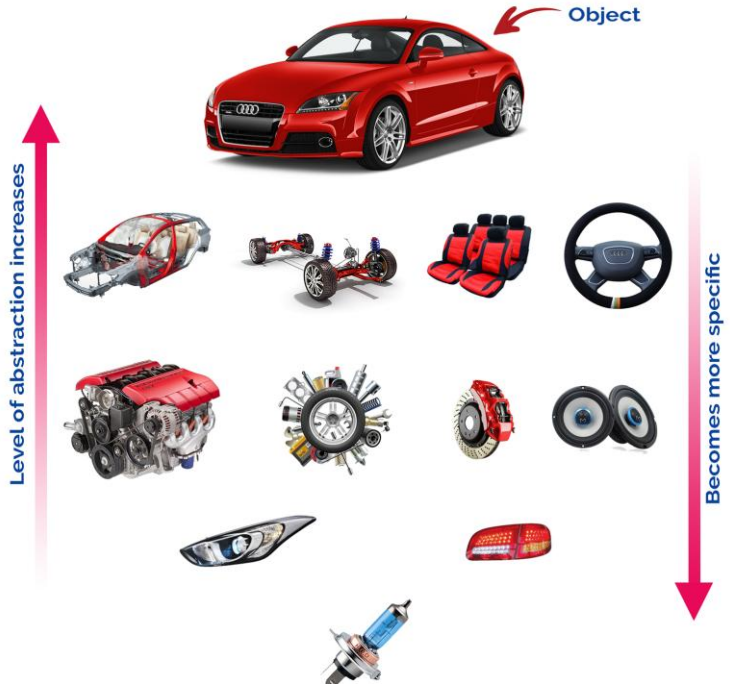
Polymorphism lets an entity take on different meanings in different contexts.

Abstraction

Data Abstraction is the property by virtue of which only the essential details are displayed to the user.

The trivial or non-essential units are not displayed to the user.

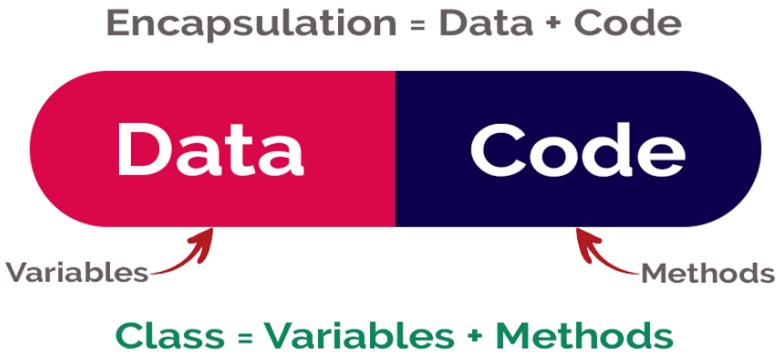
Ex: *A car is viewed as a car rather than its individual components.*



Encapsulation

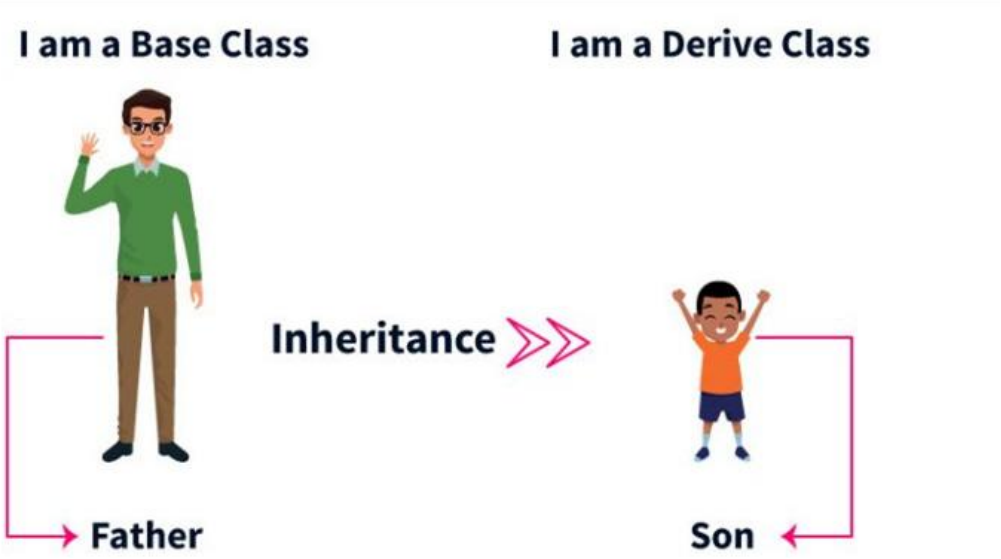
“Whatever changes, encapsulate it.”

- Encapsulation is the process of combining data and code into a single unit (object / class).
- In programming, data is defined as variables and code is defined as methods.
- The Java programming language uses the class concept to implement encapsulation.



Inheritance

- **Inheritance** is a mechanism by which one class acquires the properties and behaviors of the parent class.
- It’s essentially creating a *parent-child relationship* between classes.



Polymorphism

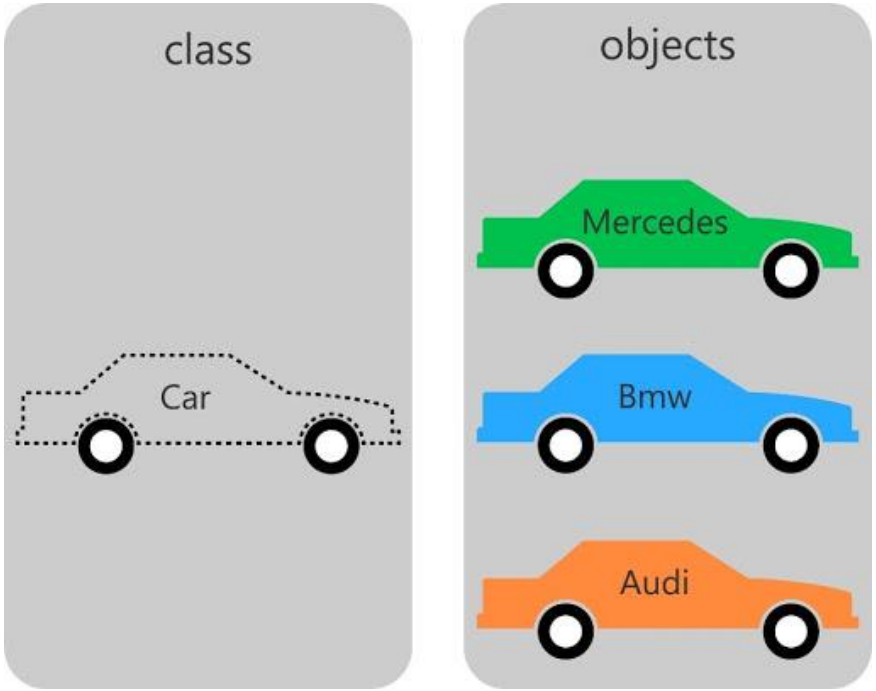
“One interface – multiple implementations.”

- **Polymorphism** means taking many forms, where ‘*poly*’ means many and ‘*morph*’ means forms. It is the ability of a variable, function or object to take on multiple forms.
- Polymorphism allows define one interface or method and have multiple implementations.

Objects and Classes

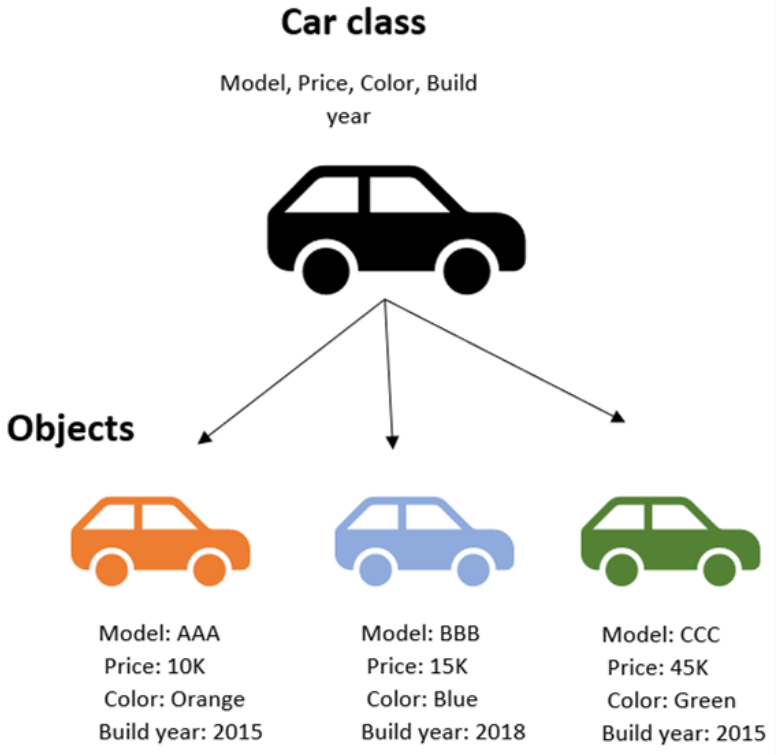
Objects and Classes

- An **object** represents an entity in the real world that can be distinctly identified.
  - **Ex**: a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
  - An object has a unique identity, state, and behaviors.
- **Classes** are constructs that define objects of the same type.



Objects

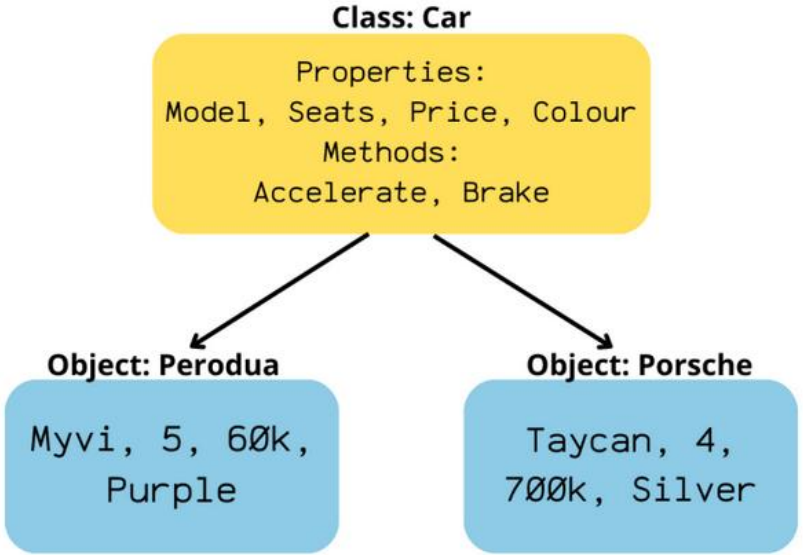
- An object has a *unique identity, state,* and *behaviors.*
- The state of an object consists of *a set of data fields* (properties) with their current values.
- The behavior of an object is defined by a set of *methods*.



Classes

A **class**:

- Is a blueprint or recipe for an object;
- Describes an object's *properties* and *behaviors*;
- The *attributes/properties* of an object are implemented using variables.
- The *behavior* of an object is implemented using methods.
- Is used to create object instances.



The Components of a Class

```java
/**
* Car class
* @author
*/
public class Car {
    String color;
    String model;
    int buildYear;
    double price;
    public void start() {
        ...
    }
    public void stop() {
        ...
    }
    public void accelerate(){
        ...
    }
}
```

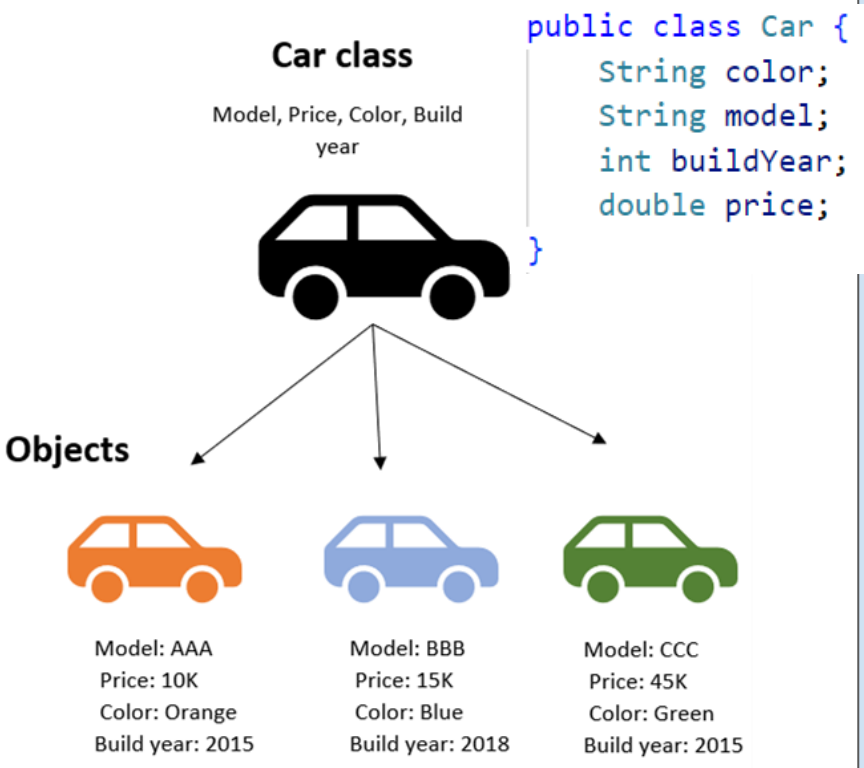
Java Doc

attributes/fields (properties)

methods (behaviors)

Instance Variables

- The variables `color`, `price`, `model` and `buildYear` are used to store the state of an object, also called an **instance**.
- They're called *instance variables* or *instance attributes*.
- Each object has its own copy of the instance variables.
- The instance variables are defined within a class but outside all methods in a class.



Variables Types Default Initialization

If uninitialized, the variables that are used as a class or instance variables get a default value:

- byte, short, int, long -> 0
- float, double -> 0.0
- boolean -> false
- char -> \u0000
- reference variables -> null

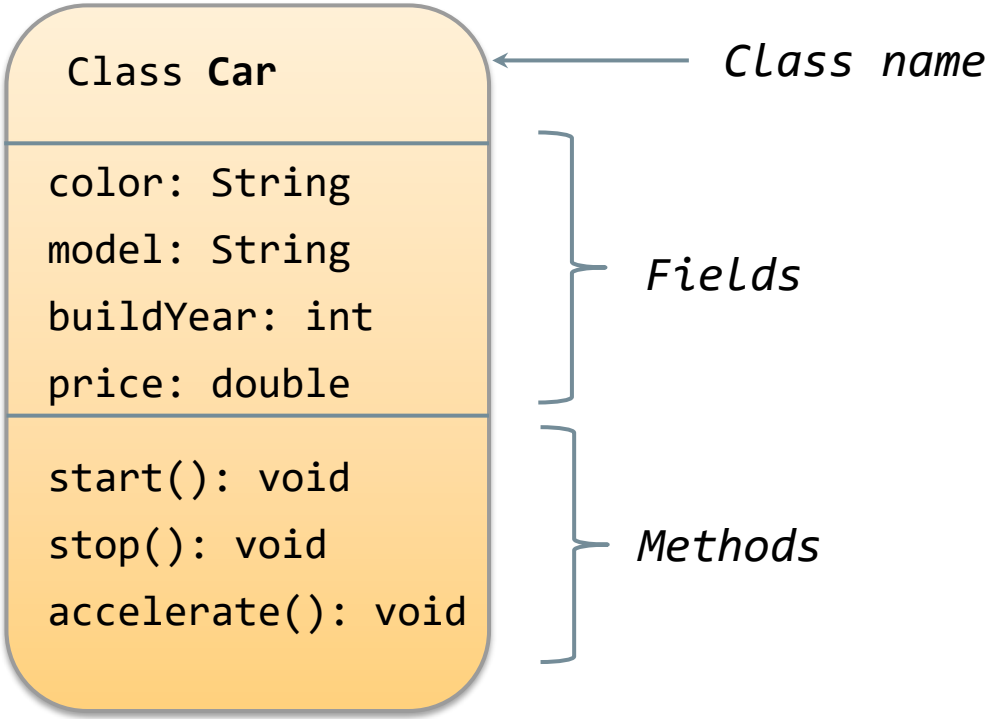
```
public class Car {
    String color;
    String model;
    int buildYear;
    double price;
}
```

Class Methods

- The methods `start()`, `stop()`, and `accelerate()` are *instance methods*.
- The methods are generally used to manipulate the instance variables.

```
public class Car {
    public void start() {...}
    public void stop() {...}
    public void accelerate(){...}
}
```

Modeling Properties and Behaviors



About UML diagrams: https://tallyfy.com/uml-diagram/

Object Syntax in Java

The syntax to create an object is:

variable becomes a reference to that object

The **new** keyword creates (instantiates) a new instance.

```
ClassName objectRefVar = new ClassName();
```

```
public static void main(String[] args) {
  Car car01 = new Car();  // Declare and instantiate
  Car car02;              // Declare the reference
  car02 = new Car();      // Then instantiate

  new Car();              // Instantiation without a reference
                          // we can not use this object later
                          // without knowing how to reference it
}
```

Class vs Object

Class	Object
Class is a blueprint or template from which objects are created.	Object is an instance of a class.
When a class is created, no memory is allocated.	Objects are allocated memory space whenever they are created.
The class has to be declared first and only once.	An object is created many times as per requirement.
A class is a logical entity.	An object is a physical entity.
It is declared with the class keyword like `class Car{}`	It is created with the new keyword like `Car car01 = new Car();`
We cannot manipulate class as it is not available in memory.	Objects can be manipulated.
Example: *Car* is a class.	If Car is the class then *volvo*, *bmw*, *porche*, *toyota* are its objects which have different properties and behaviors.

Constructors

Constructors

- A **constructor** in Java is a block of code similar to a method that's called when an instance of an object is created.
- Constructors must have the same name as the class itself.
- Constructors do not have a return type (Not even void).
- Constructors are invoked using the new operator when an object is created.
- Constructors play the role of initializing objects.
- The constructor is the first method called when an object is instantiated. Its purpose is primarily to set default values.

Constructors

```
class Car {
    ...
    /*
        * Constructor of class Car
    */
    Car(){
        //constructor body
    }
}
```

- If we do not create a constructor of a class, Java creates a default constructor with data members which has values like zero, null, etc.

Accessing instance variables & methods

- Use the object member access operator
  - Also called the dot operator (.)
- Reference the object's data using `objectRefVar.data`
- Invoke the object's method using `objectRefVar.methodName(arguments)`

Creating objects and Accessing instance variables & methods

```java
package oop;

public class Car {
 String model;
 double price;
 String color;
 int buildYear;
 void accelerate(int acceleration){
   System.out.println("->"+acceleration+" km/h");
 };
 void brake(){
   System.out.println("1ms");
 };
 public Car() {
 }
 public Car(String model, String color, int buildYear) {
   this.model = model;
   this.color = color;
   this.buildYear = buildYear;
 }

 @Override
 public String toString() {
   return "Car{" +
          "model='" + model + '\'' +
          ", price=" + price +
          ", color='" + color + '\'' +
          ", buildYear=" + buildYear +
          '}';
 }
}
```

```java
package oop;
import java.util.*;
public class Objects {
 public static void main(String[] args) {
   Car porsche=new Car(),perodua;
   String model, color;
   int buildYear;
   porsche.model="Taycan";
   porsche.color="black";
   porsche.buildYear=2020;
   int acceleration=250;
   System.out.print(porsche.toString());
   porsche.accelerate(acceleration);
   //Use constuctor with parameters
   Scanner scanner=new Scanner(System.in);
   System.out.println("Input Perodua model ");
   model=scanner.nextLine();
   System.out.println("Input Perodua color ");
   color=scanner.nextLine();
   System.out.println("Input Perodua buld year ");
   buildYear=scanner.nextInt();
   perodua=new Car(model,color,buildYear);
   System.out.println("Input acceleration ");
   acceleration=scanner.nextInt();
   System.out.print(perodua.toString());
   perodua.accelerate(acceleration);
 }
}
```

Object References

Working with Object References

The TV is like the object that is accessed using a reference.

The remote is like the reference used to access the camera.

Working with Object References

remote1

remote2

```
TV remote1 = new TV();
TV remote2 = new TV();
remote1.play();
remote2.play();
```

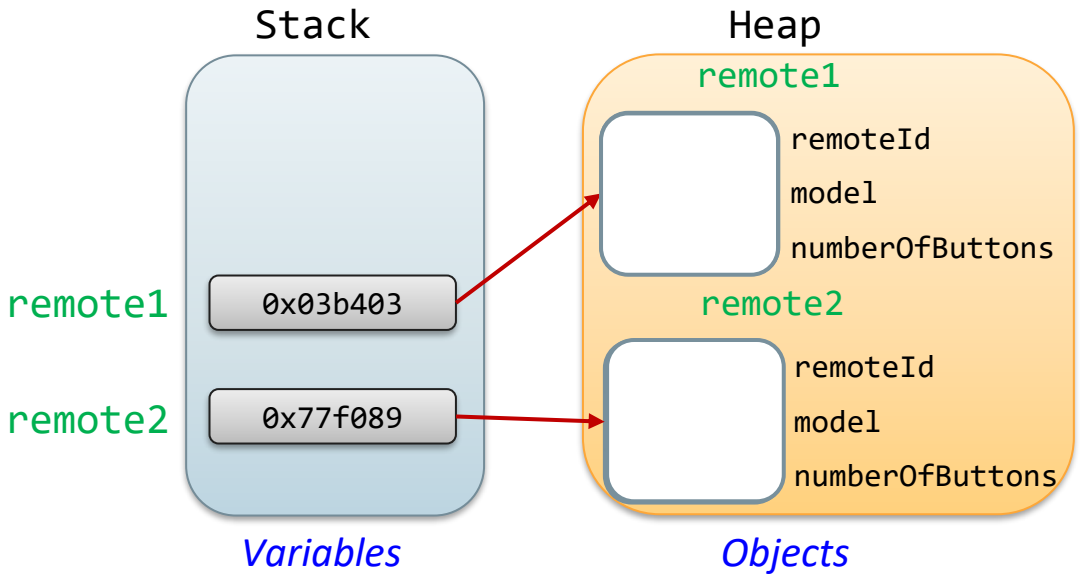
There are two TV objects

Working with Object References

```
TV remote1 = new TV();
TV remote2 = new TV();

remote1.remoteId = 1;
remote1.model = "Samsung";
remote1.numberOfButtons = 14;

remote2.remoteId = 2;
remote2.model = "LG";
remote2.numberOfButtons = 18;
```



The **stack** holds local variables, either primitives or reference types.

The heap holds objects.

Working with Object References


```
TV remote1 = new TV();
TV remote2 = remote1;
remote1.play();
remote2.stop();
```

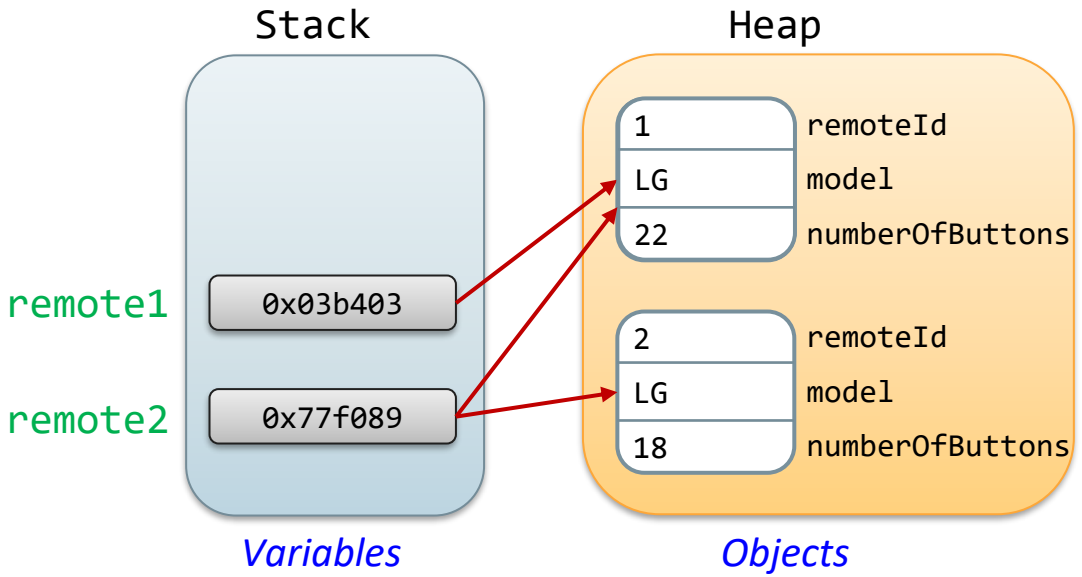
There is only one TV object

Working with Object References

```
...

TV remote2 = remote1;

remote1.model = "LG";
remote2.numberOfButtons = 22;
```



The **stack** holds local variables, either primitives or reference types.

The heap holds objects.

Thank you for your attention!