JAVA OOPS

INTRODUCTION TO OO PROGRAMMING IN JAVA

KavinSchool

JAVA OOPS

1

INTRODUCTION TO

OBJECT ORIENTED SOFTWARE DEVELOPMENT

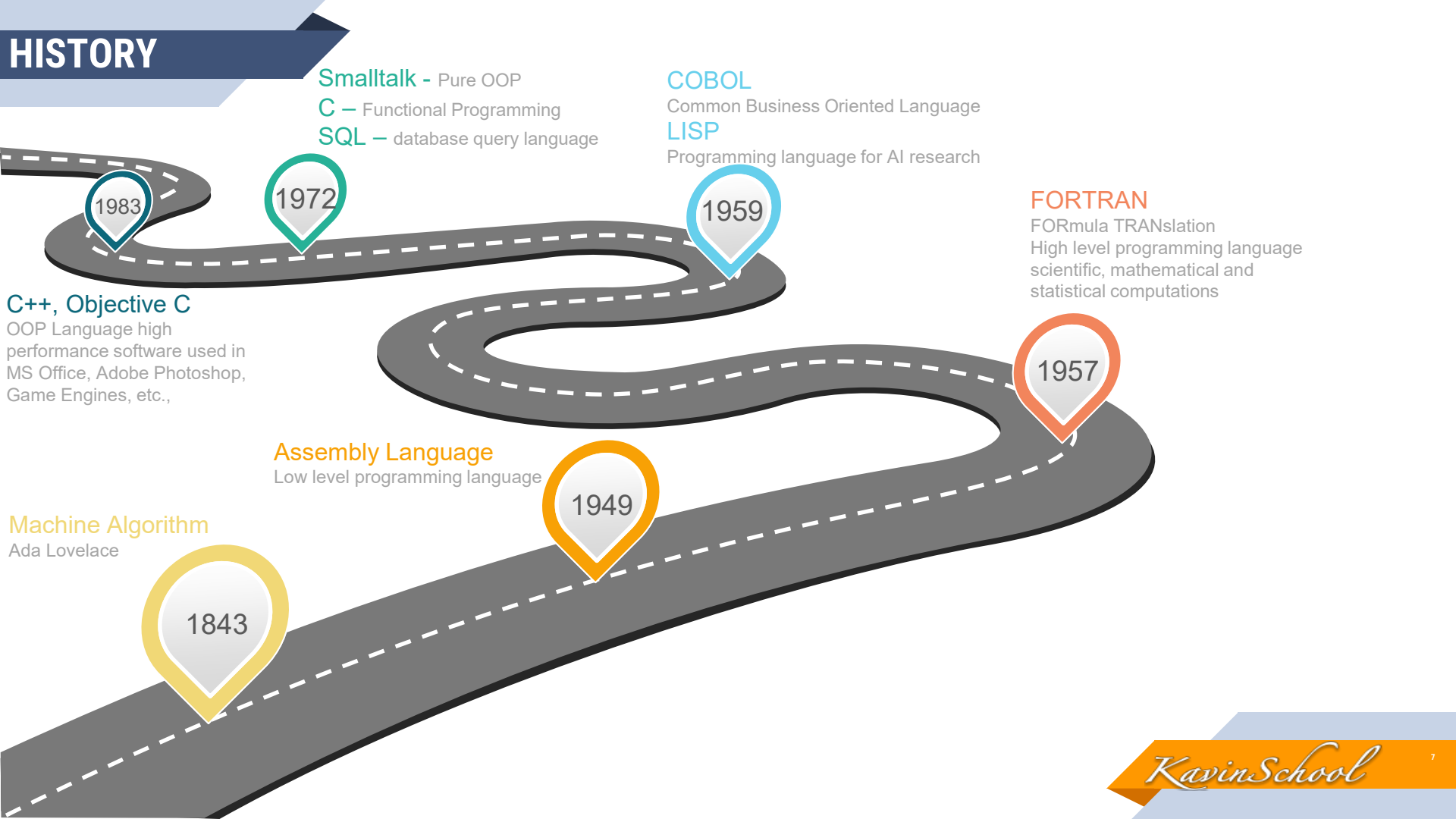
Learning Objectives

- What is Object Oriented Programming?
- History, Evolution, and Motivations for OO
- Designing software with an OO approach

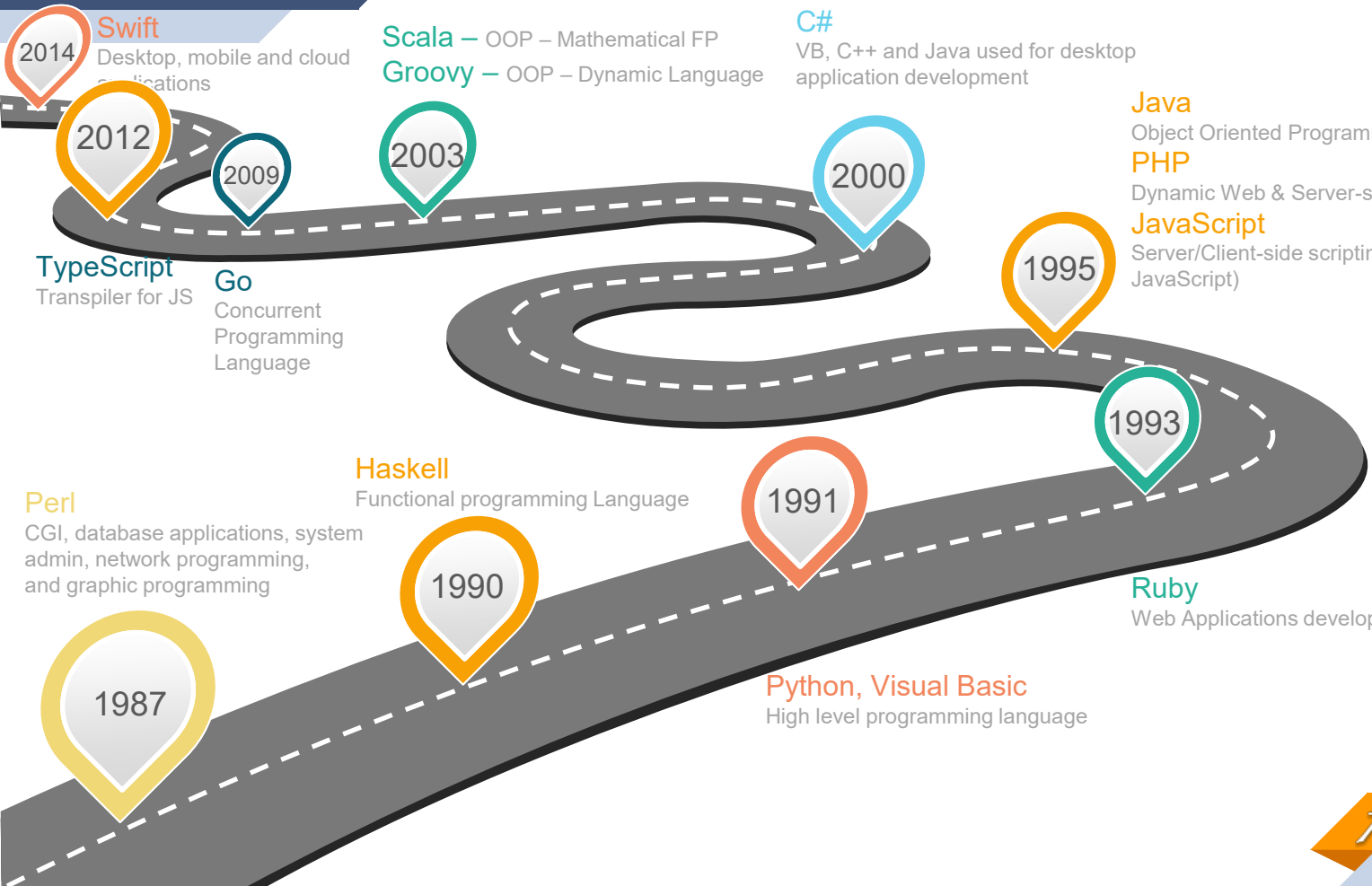
WHAT IS OOP?

- Object Oriented programming (OOP) is a programming paradigm that relies on the concept of classes and objects
- OOP began with the **Simula** language (1967), which added information hiding to **ALGOL** (Algorithmic Language) originally developed in 1958
- Another influential OO language was **Smalltalk** (1980), in which a program was a set of objects that interacted by sending messages to one another

HISTORY


Machine Algorithm
Ada Lovelace
1843
Assembly Language
Low level programming language
1949
FORTRAN
FORmula TRANslation
High level programming language
scientific, mathematical and
statistical computations
1957
COBOL
Common Business Oriented Language
LISP
Programming language for AI research
1959
Smalltalk - Pure OOP
C – Functional Programming
SQL – database query language
1972
C++, Objective C
OOP Language high
performance software used in
MS Office, Adobe Photoshop,
Game Engines, etc.,
1983

HISTORY


2014
Swift
Desktop, mobile and cloud
2012
2009
TypeScript
Transpiler for JS
Go
Concurrent
Programming
Language
2003
Scala – OOP – Mathematical FP
Groovy – OOP – Dynamic Language
2000
C#
VB, C++ and Java used for desktop
application development
1995
Java
Object Oriented Programming language
PHP
Dynamic Web & Server-side development language
JavaScript
Server/Client-side scripting language (Every web-site uses
JavaScript)
1993
Ruby
Web Applications development language
1991
Haskell
Functional programming Language
1990
Perl
CGI, database applications, system
admin, network programming,
and graphic programming
1987
Python, Visual Basic
High level programming language

OOPS

Object Oriented Programming System

WHY OOP?

Improved Software Quality

Real world modeling and relationships

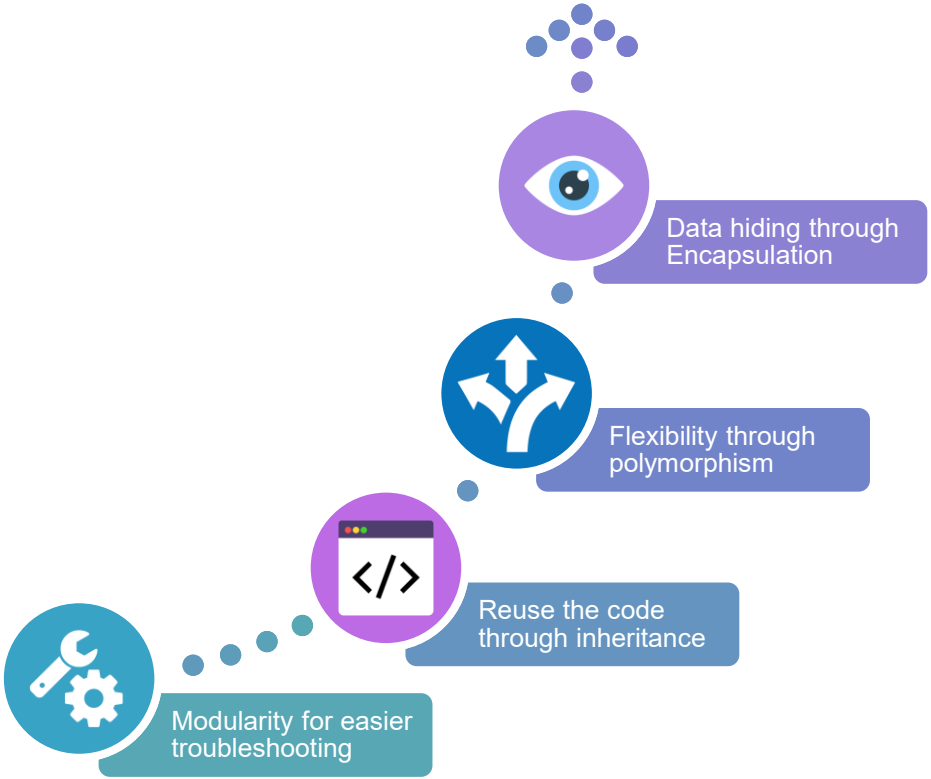
Lower Cost of Development

Faster Development

Higher Quality Software

Improved Software Maintainability

BENEFITS OF OOP



> *OOAD is a popular technical approach for analyzing and designing an application system, by applying OOP, as well as visual modeling throughout the SDLC to promote better communication and product quality*

> *OOAD models and designs a system as a group of interacting objects. **Object** is the term used to describe some entity or "**thing**" of interest.*
>
> *These objects are typically modeled after*
>
> *real world entities or concepts*

WHAT IS OOAD?

Object Oriented Analysis

Object Oriented Design

ANALYSIS VS DESIGN

Analysis

object-oriented analysis phase of software development, the system requirements are determined, the classes are identified and the relationships among classes are identified

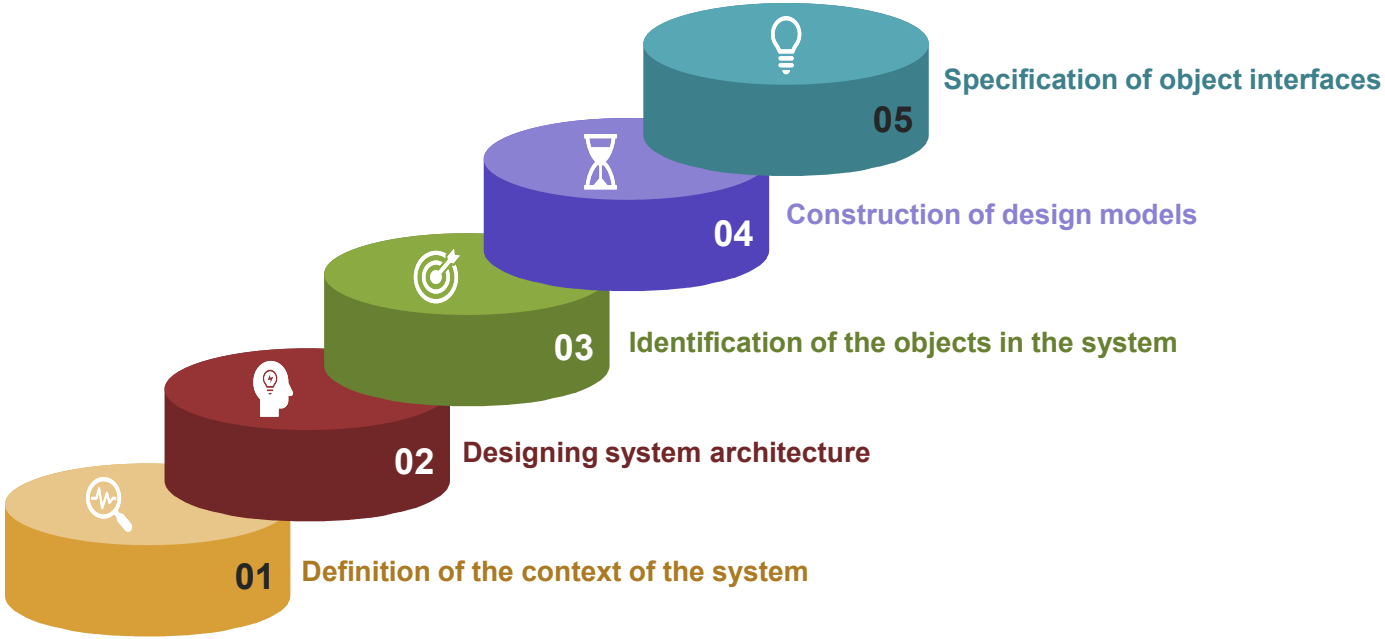
Design

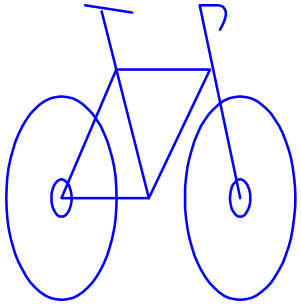
object-oriented design phase of software development, a detailed description is constructed specifying how the system is to be built on concrete technologies

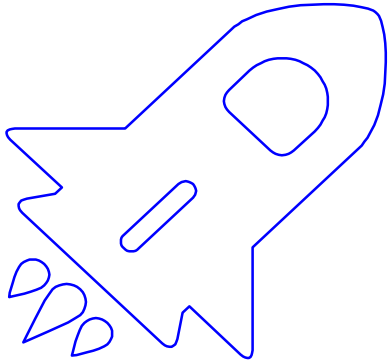
DURING ANALYSIS

1. Identify objects and group into classes
2. Identify the relationships among classes
3. Create user object model diagram
4. Define user object attributes
5. Define the operations that should be performed on the classes

DURING DESIGN







Designing software with an OO approach

> *Objects have attributes which can be set to specific values. This defines the state of the object. Objects also have methods or functions which define their behavior.*

REAL WORLD EXAMPLE - CAR

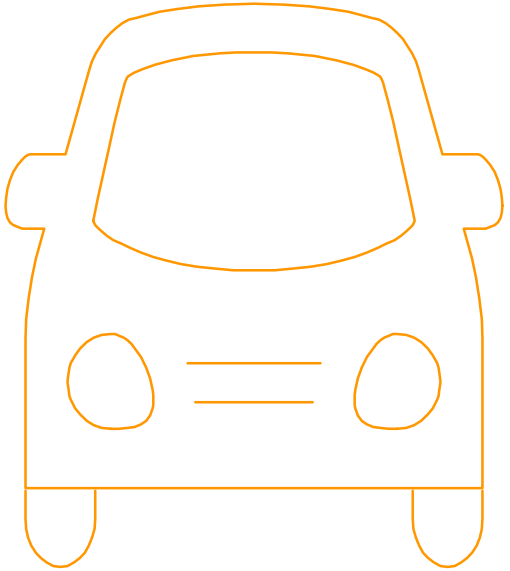
Consider the real world object “Car”.

Car has attributes that can be defined with specific values:

make = toyota

model = corolla

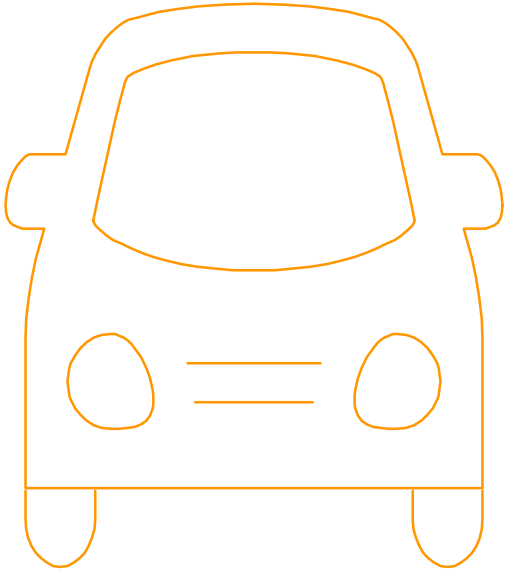
year = 2019



REAL WORLD EXAMPLE - CAR'S ATTRIBUTES

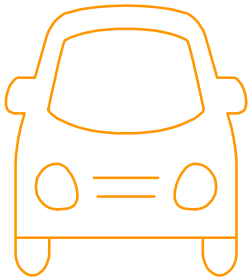
"Car" - attributes

- Color = red
- maximum speed = 180 mph
- current speed = 80 mph
- ideal tire pressure = 35 psi
- current tire pressure = 31 psi
- remaining fuel = 32 gallons



ATTRIBUTES DEFINE “STATE” OF THE OBJECT

Car’s attributes define the state of the vehicle

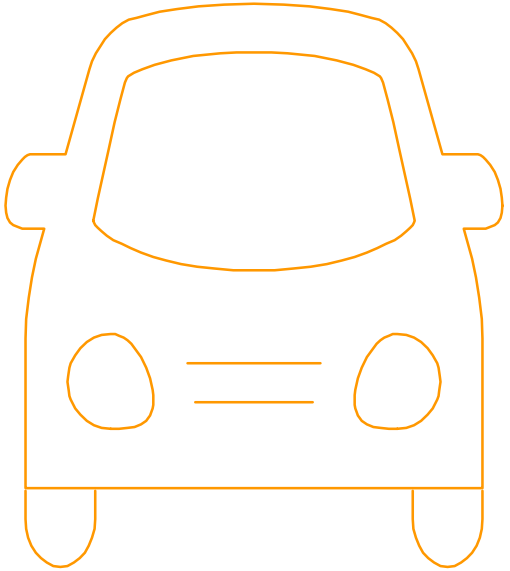


REAL WORLD EXAMPLE - CAR'S BEHAVIORS

- Lights
  - On
  - Off
- Wiper
  - On
  - Off
- Break
  - On
  - Off

"Car" - Methods

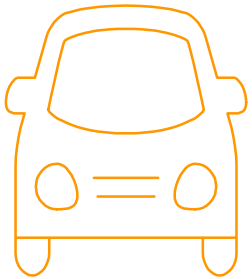
- Start
- Drive
- Park
- Accelerate
- Fill Tires
- Refuel
- Set Heater



> *Behaviors of the real world object can be represented as a method of the object when designing the system. These methods can change the values of the attributes causing a change in state*

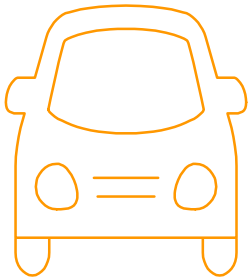
OOA focuses on **What** the system does it?

Static Structure and Behavior



OOD focuses on **How** the system does it?

Run time implementation



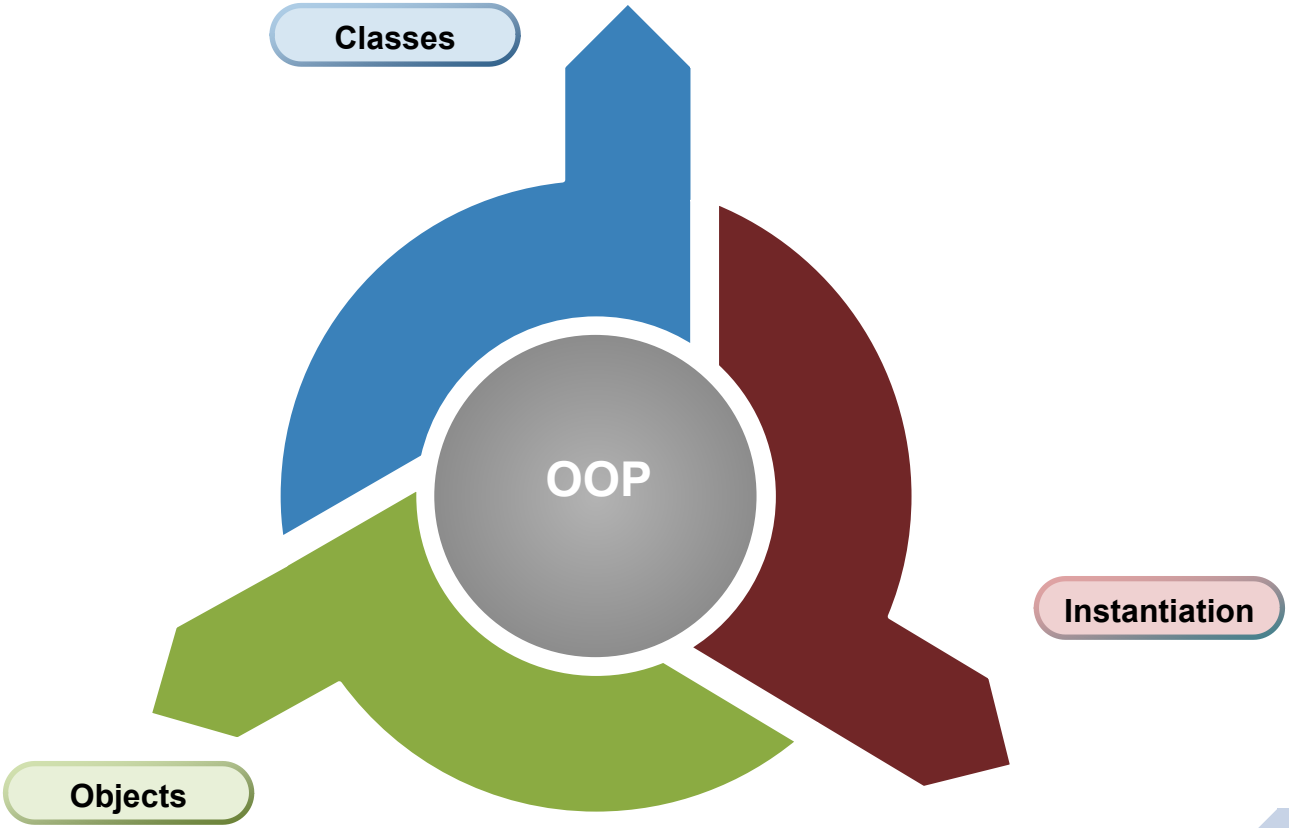
2

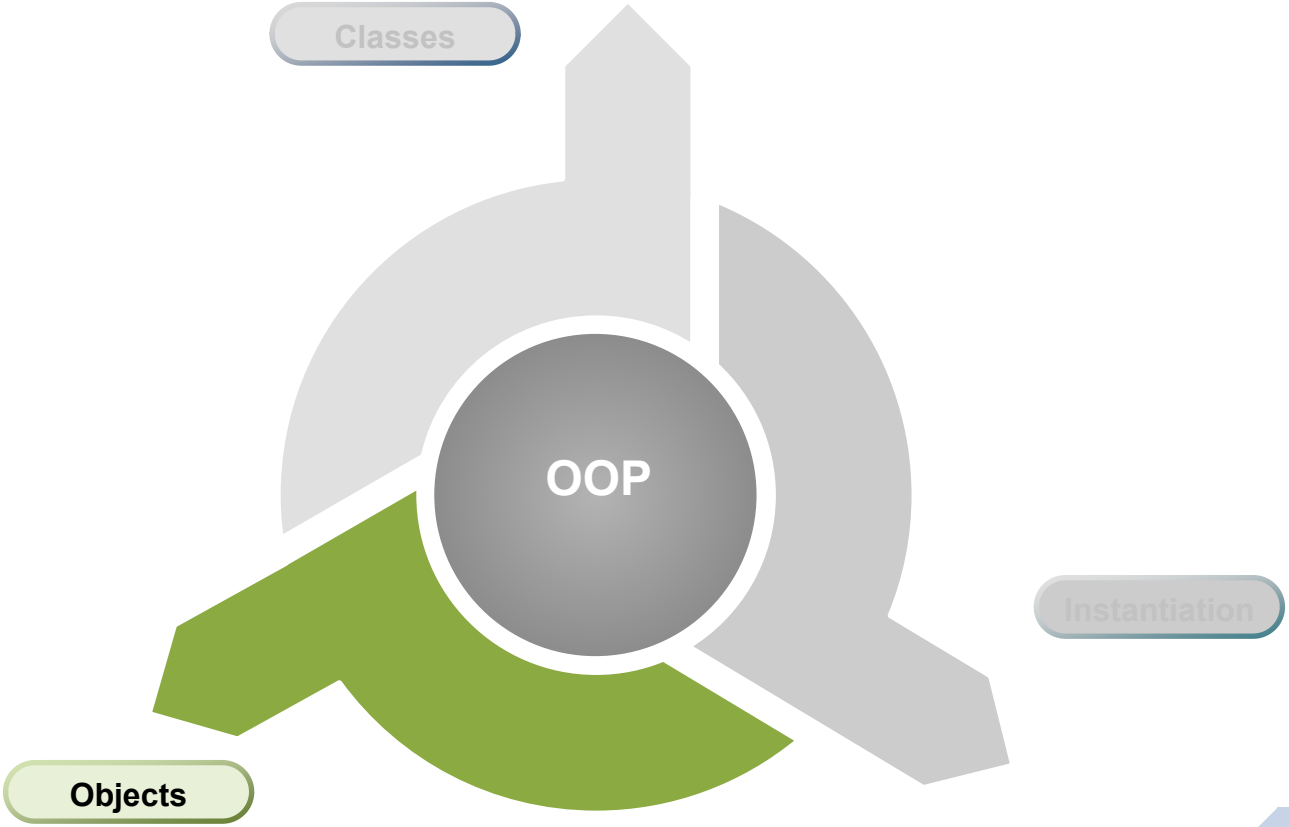
CORE CONCEPTS

OBJECT ORIENTED SOFTWARE DEVELOPMENT

Learning Objectives

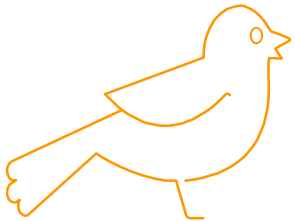
- Classes, Objects, Instantiation
- States / Behaviors, Properties / Methods

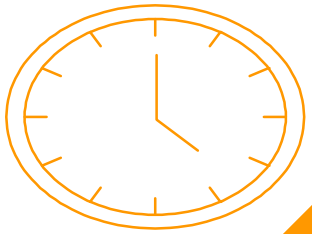




Objects - things people can observe and interact with

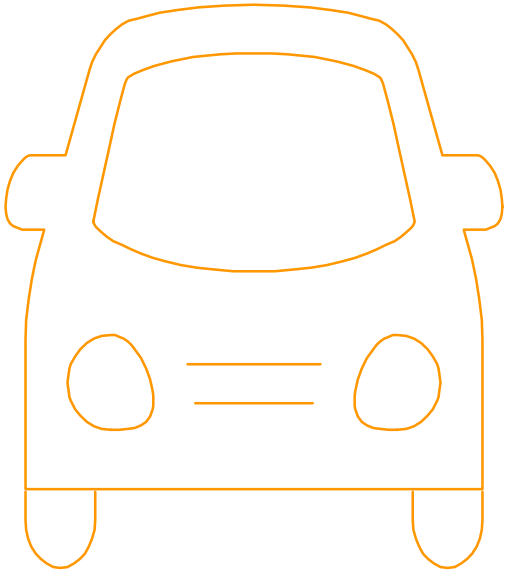
Car, bird, house, clock



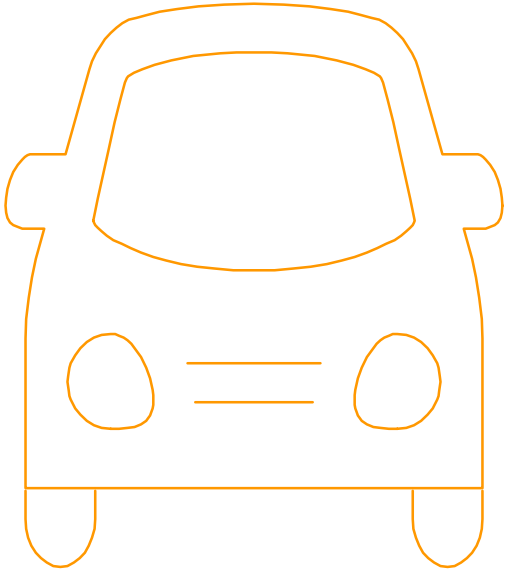
OBJECTS IN JAVA

- all cars have wheels, but not all cars have the same number of wheels



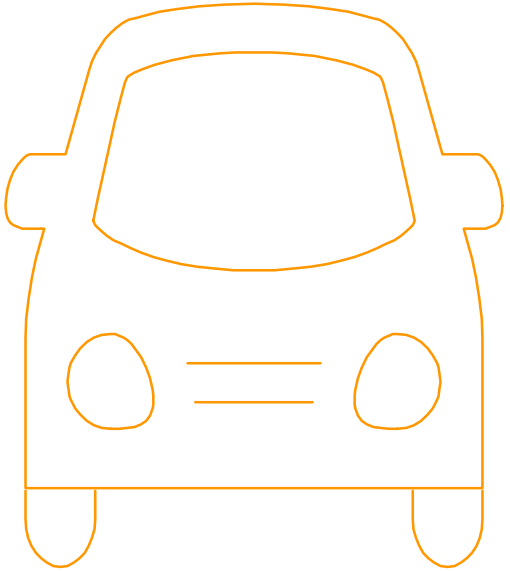
OBJECTS ARE REAL WORLD MODEL

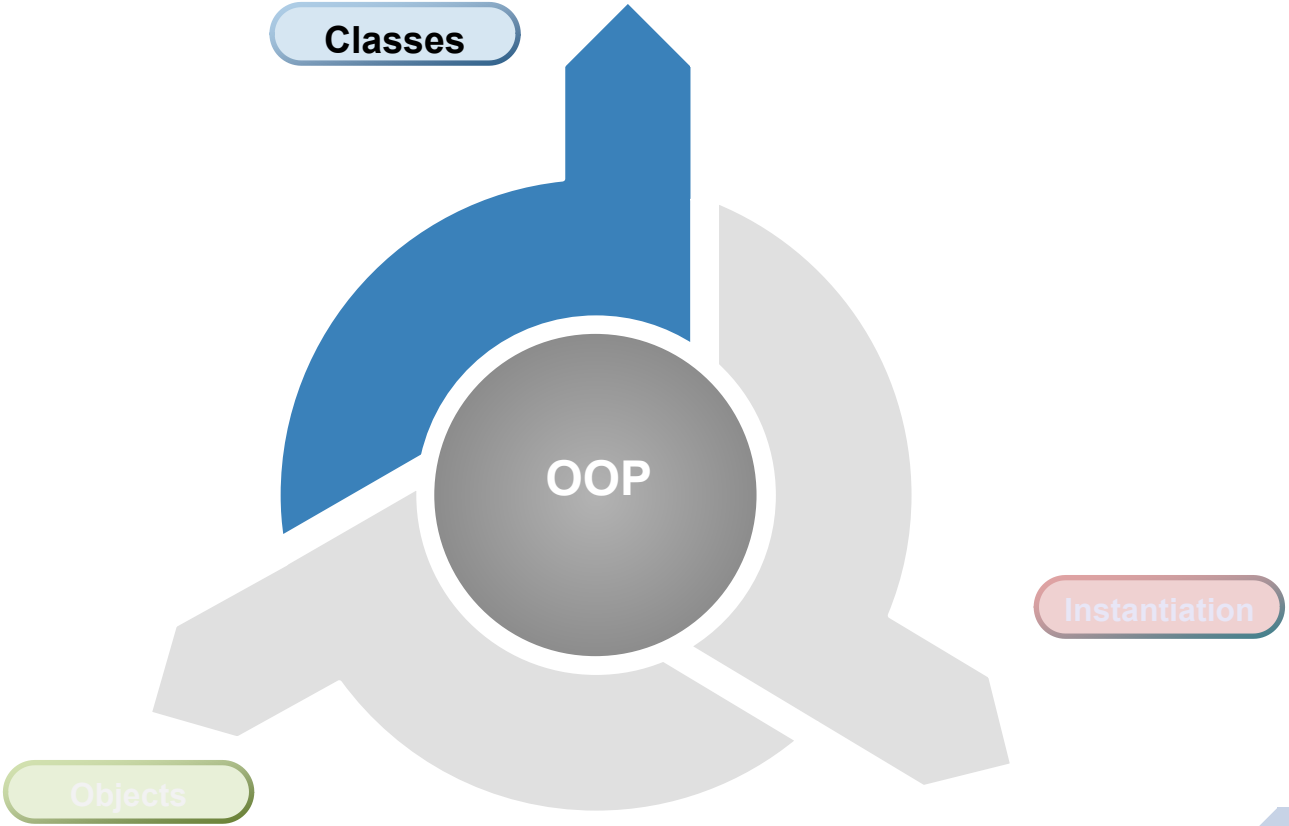
- Objects in Java are used to model real-world objects, giving them properties and behavior just like their real-world counterparts



PROPERTIES

- The properties what makes up an object
- Similar Objects share the same properties
- May have different states/values for those properties





CLASSES

- A Class is a template of an object
- A Class definition contains attributes and behaviors
- Attributes holds the state information of the object
- Variables are used to define the attributes
- Behaviors are activities that is visible outside of the object
- Methods are used to define the behavior of an object

CLASSES

Classes are declared using class keyword

Classes attributes are static in nature. Shared by all the objects of a class

Object attributes are called instance variables

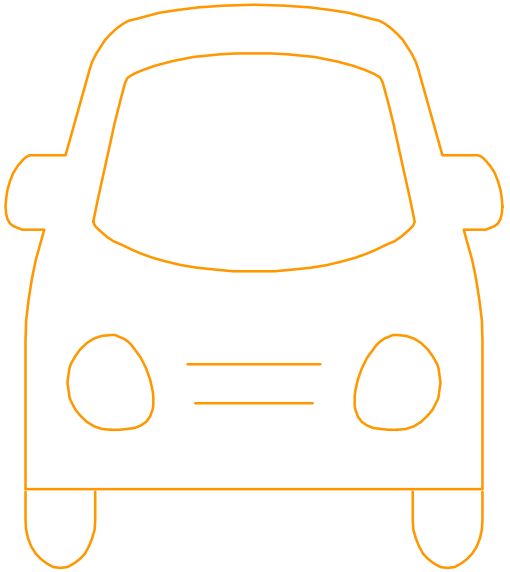
Object behaviors are called methods

Syntax

```
accessSpecifier Modifier class className extends
superClassName  implements interfaceName {
static classVariable 1; //optional
static classVariable2;
static classVariableN;

type instanceVariable 1; //optional
type instanceVariable 2;
type instanceVariable N;

type methodName1(paraVar1, paraVar2,ParaVar3) {
 Statements; //Optional }

type methodName2(paraVar1, paraVar2,ParaVar3) {
 Statements; //Optional}

type methodName2(paraVar1, paraVar2,ParaVar3) {
 Statements; //Optional}
}
```

EXAMPLE OF AN OBJECT

```
public class Car {
    private String make;
    private String model;
    private int year;

}
```



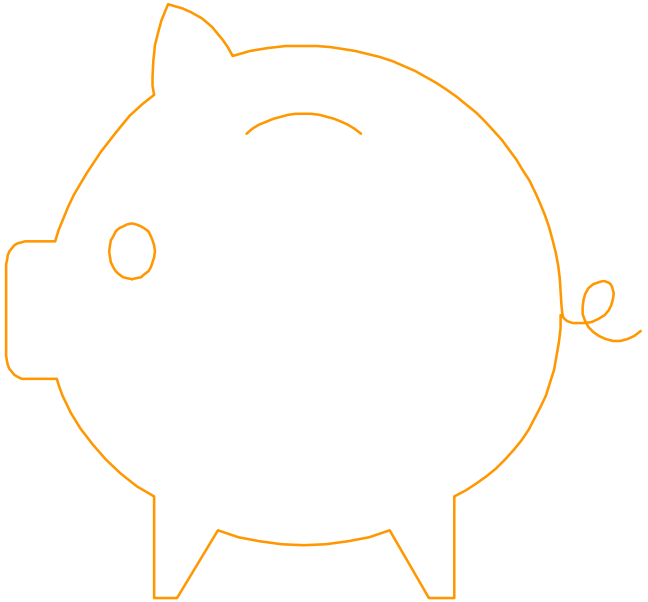
OBJECTS


TOBACCO TIN
SUPER SOFT ICE
HOT DOGS
HOT DOGS ICE CREAMS
LONDON
A-Z

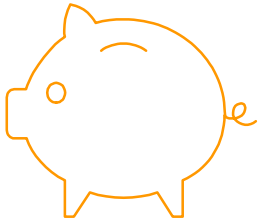
EXERCISE

Question:

Create a **Pig** class with **name**, **age**, and **color** properties?



PIG OBJECT



```
public class Pig {
    private String name;
    private int age;
    private String color;

}
```

SETTERS AND GETTERS IN OBJECTS

Getter methods allows you to get the value of an instance

Setter methods allows you to change an existing value of an object

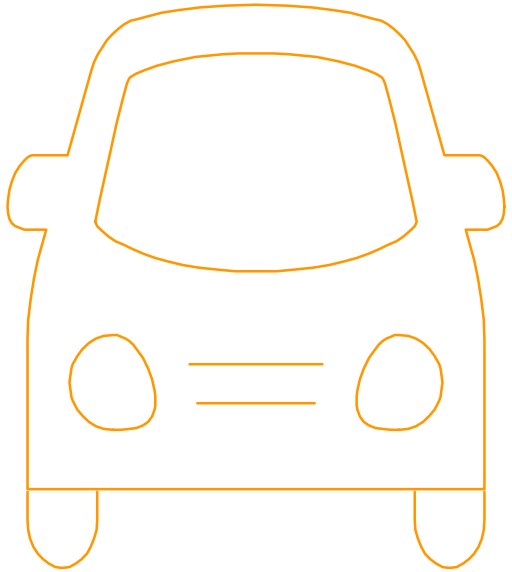
Getters and Setters restricts how the objects values can be accessed

WHY YOU NEED GETTERS AND SETTERS

- It gives simpler syntax
- It allows equal syntax for properties and methods
- It can secure better data quality
- It is useful for doing things behind-the-scenes

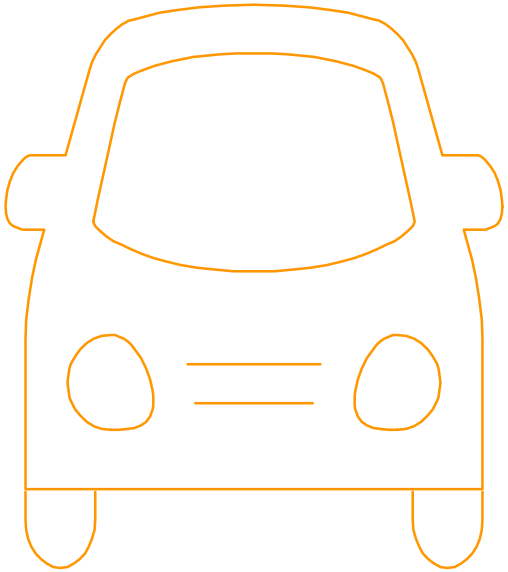
GETTER METHODS

```
public String getMake() {
    return make;
}

public String getModel() {
    return model;
}

public int getYear() {
    return year;
}
```



SETTER METHODS

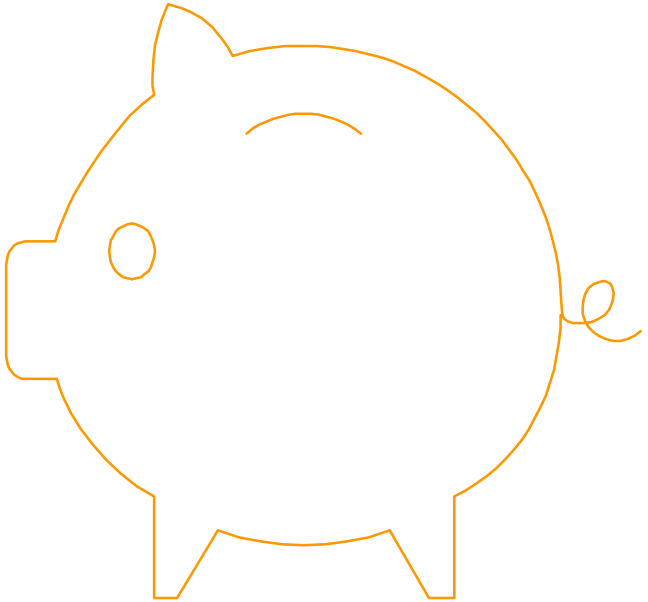
```
public void setMake(String make) {
    this.make = make;
}

public void setModel(String model) {
    this.model = model;
}

public void setYear(int year) {
    this.year = year;
}
```



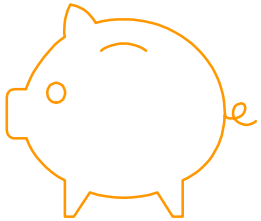
EXERCISE

Question:

In the pig class create **setter** and **getter** methods for **name**, **age** and **color**.

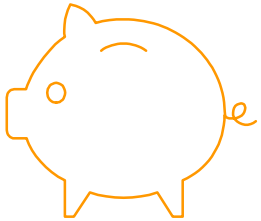


PIG SET-METHODS



```
public void setName(String name) {
    this.name = name;
}

public void setAge(int age) {
    this.age = age;
}

public void setColor(String color) {
    this.color = color;
}
```

PIG GET-METHODS



```
public class Pig {
    public String getName() {
       return name;
    }
    public int getAge() {
       return age;
    }
   public String getColor() {
      return color;
   }
}
```

METHODS

- A function (subroutine , method, procedure, or subprogram) is a portion of code within a larger program, which performs a specific task and can be relatively independent of the remaining code
- A function can return a value or type void
- A function definition consists of the function name, followed by

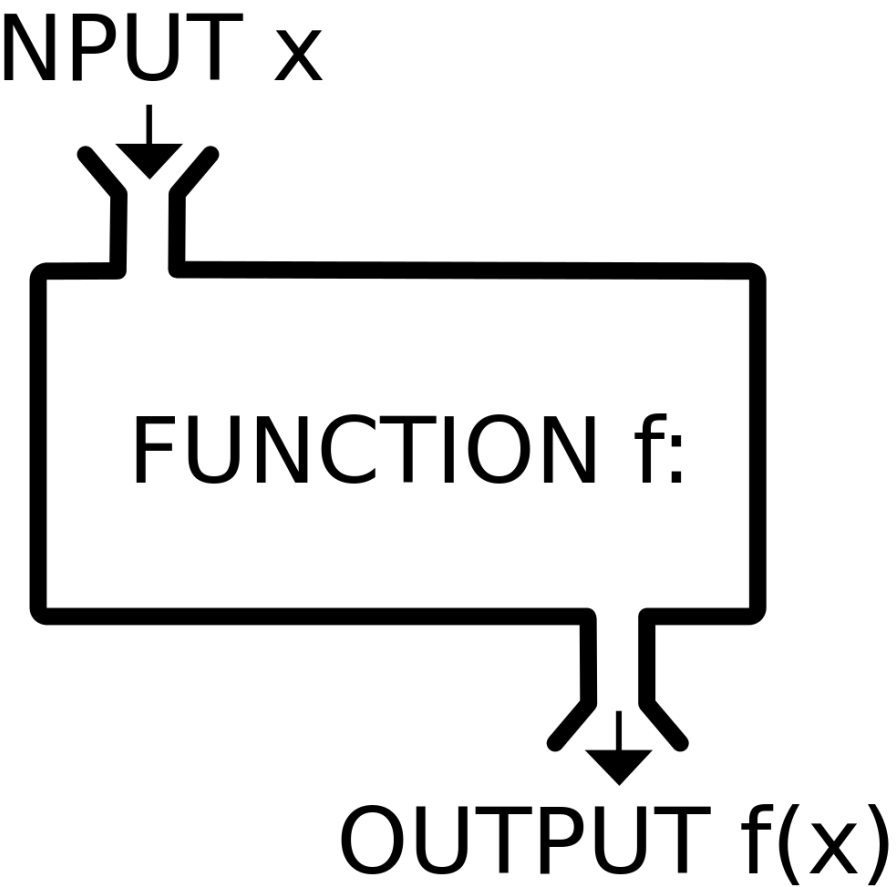
A method in Java is defined with a return type when nothing is returned then it is called a void return type

The name of the function.

A list of arguments to the function, enclosed in parentheses and separated by commas.

FUNCTION

$f(x)$

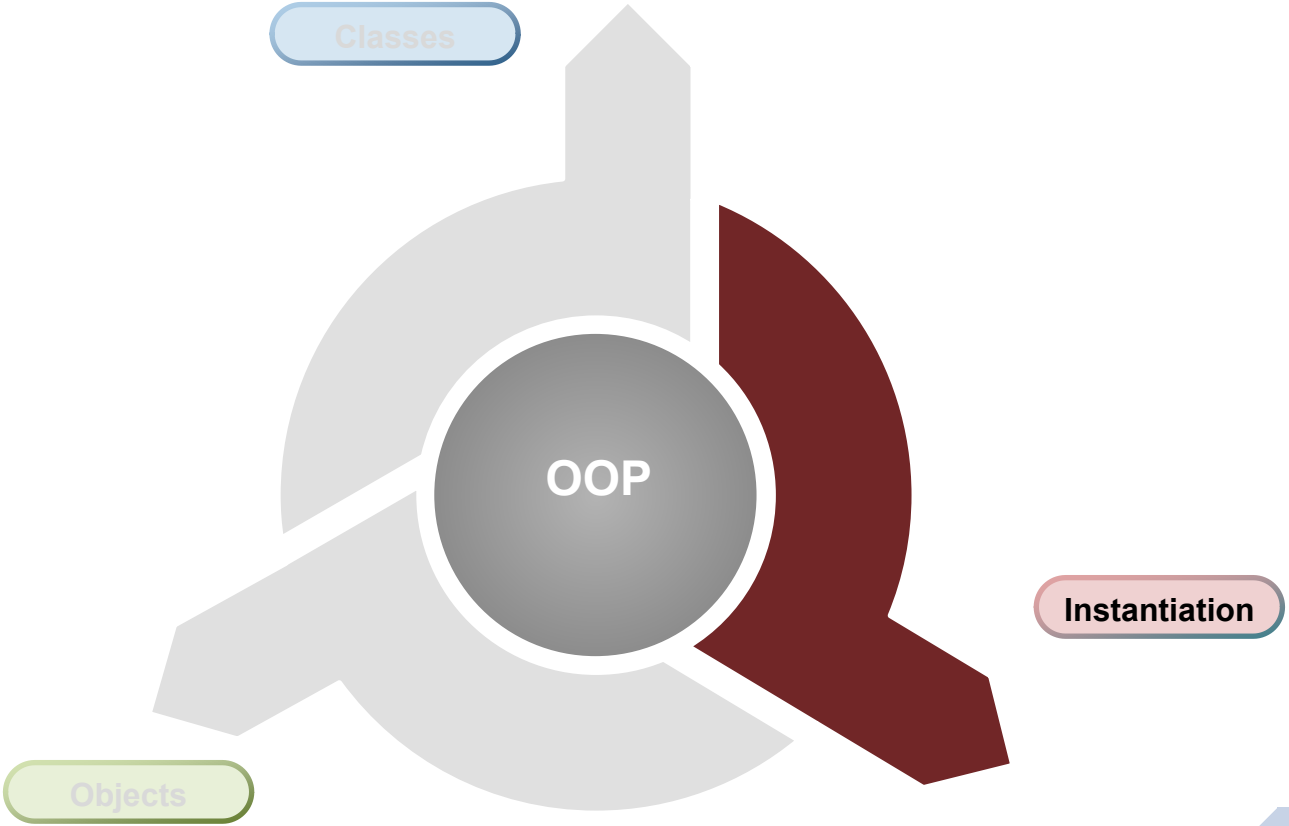


METHODS

- Methods can receive any number of arguments.
- In Java, It’s necessary that the types are explicitly defined when defining the arguments.
- Modifiers such as public, private, and protected can be added.
- By default, if no visibility modifier is provided, the method is public.

OBJECTS/INSTANCE

- An Object is a collection of attributes and related methods
- An object is an entity that has the attributes, behavior and identity
- Objects are instance of a class
- Object’s attributes and behaviors are defined in a class
- Based on a single class, you can create one or many objects (instances)
- Objects exist only when the program is running
- Objects and instance means the same



CREATING/INSTANTIATING OBJECTS

The phrase "instantiating a class" means the same thing as "creating an object"

When you create an object, you are creating an "instance" of a class, therefore "instantiating" a class

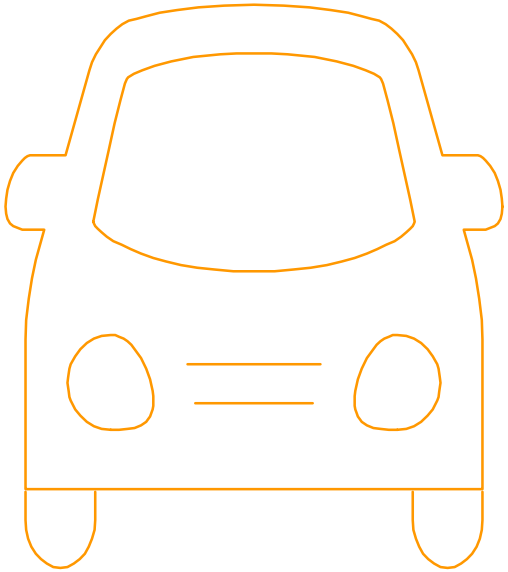
You create an object from a class using the **new** operator

The **new** operator instantiates a class by allocating memory for a new object and returning a reference to that memory

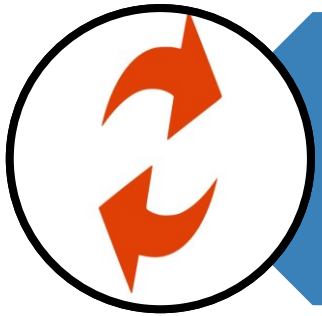
The **new** operator also invokes the object constructor

CREATING A CAR OBJECT

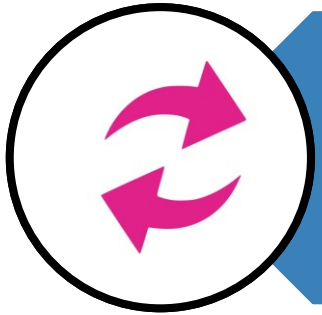
```
public class CreateCars {
    public static void main(String[] args) {
        Car toyota = new Car();
    }
}
```



SETTING A VALUE OF AN OBJECT



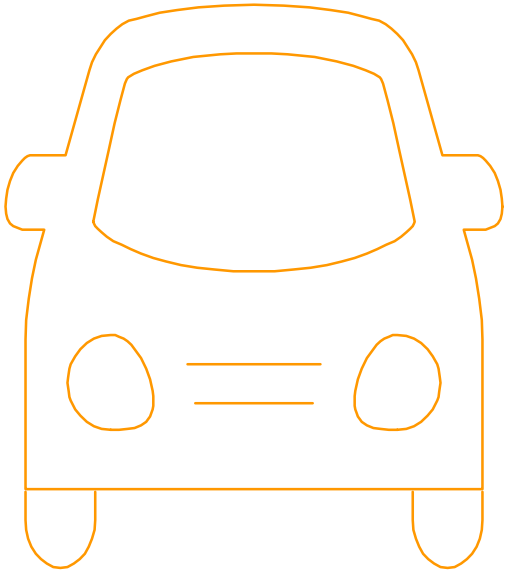
Using the setter methods of an object to set or update the values of an object



You can also use constructors to initialize and object's initial state while creating an object

INITIALIZE AN OBJECT

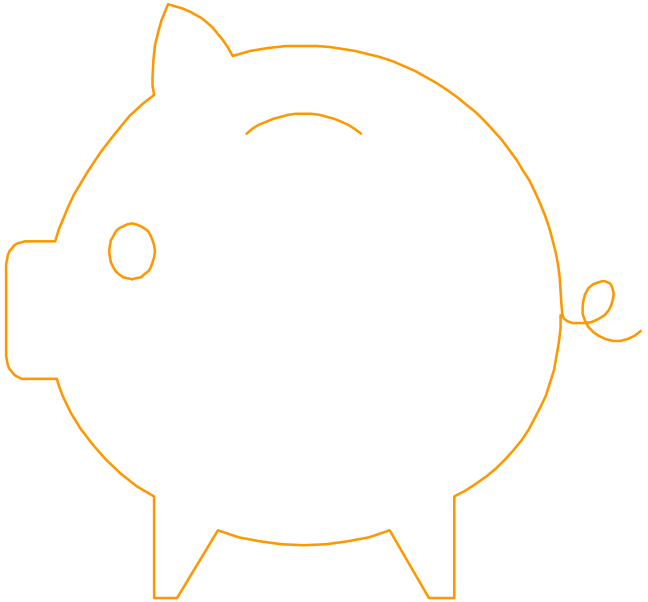
```
public class CreateCars {
    public static void main(String[] args) {
        Car toyota = new Car();
        toyota.setMake("Toyota");
        toyota.setModel("Corolla");
        toyota.setYear(2022);
    }
}
```



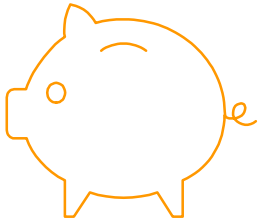
EXERCISE

Question:

Add a new **CreatePigs** class, in the java **main** method create a **Pig** object named **roseyPig** and set the value of the pig - name(“rosy”), color(“Red”) and age(4).



PIG – CREATE PIGS



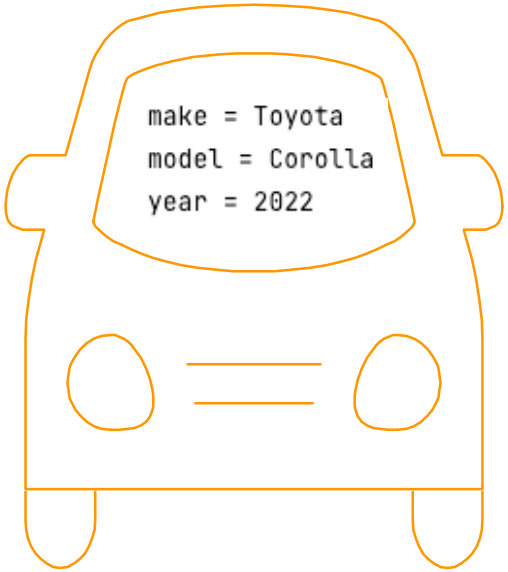
```
public class CreatePigs {
    public static void main(String[] args) {
        Pig rosyPig = new Pig();
        rosyPig.setName("Rosy");
        rosyPig.setColor("Red");
        rosyPig.setAge(4);
    }
}
```

GETTING VALUE OUT OF AN OBJECT

Using the getter methods of an object one can get the values of an object

INITIALIZE AN OBJECT

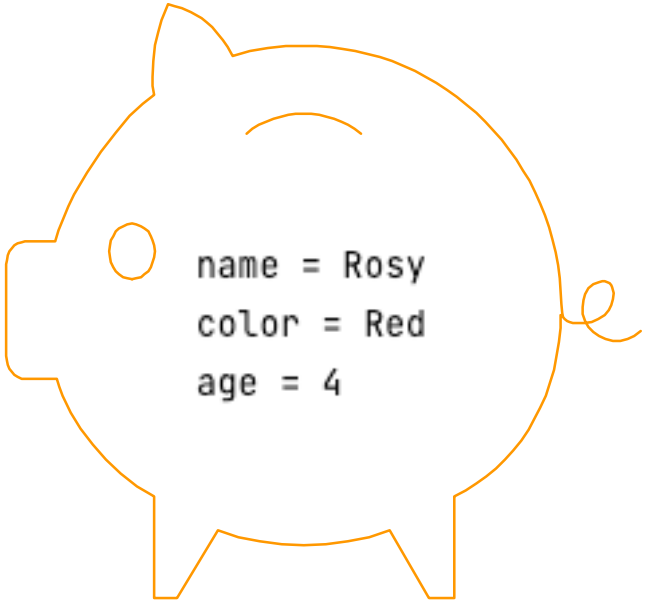
```
public class CreateCars {
    public static void main(String[] args) {
        Car toyota = new Car();
        toyota.setMake("Toyota");
        toyota.setModel("Corolla");
        toyota.setYear(2022);
        System.out.println("make = " + toyota.getMake());
        System.out.println("model = " + toyota.getModel());
        System.out.println("year = " + toyota.getYear());
    }
}
```



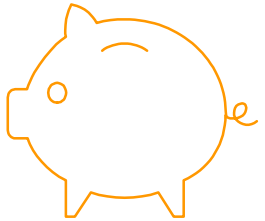
EXERCISE

Question:

In the **CreatePigs** class, **main** method print the values of the **rosyPig** object using **getName()**, **getColor()** and **getAge()** methods.



PIG GET-METHODS



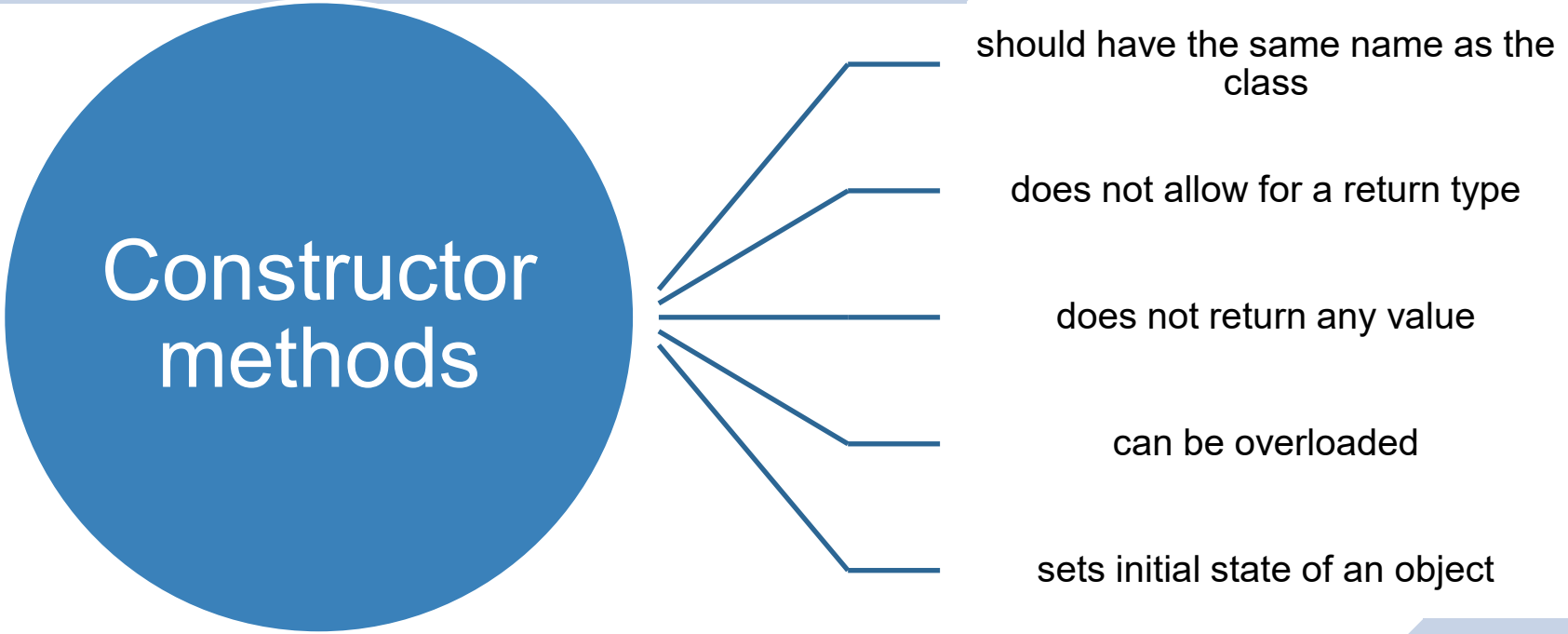
```
public class CreatePigs {
    public static void main(String[] args) {
        Pig rosyPig = new Pig();
        rosyPig.setName("Rosy");
        rosyPig.setColor("Red");
        rosyPig.setAge(4);
        System.out.println("name = " + rosyPig.getName());
        System.out.println("color = " + rosyPig.getColor());
        System.out.println("age = " + rosyPig.getAge());
    }
}
```

CONSTRUCTOR METHODS

A method which is called when an object is instantiated or constructed

A constructor cannot be called directly

CONSTRUCTOR METHODS



new KEYWORD

A **new** keyword is used to create an instance of a class which

- Allocates memory for the object
- Initializes that object's instance variables, either to initial values or to a default value
- Calls the matching constructor method of the class

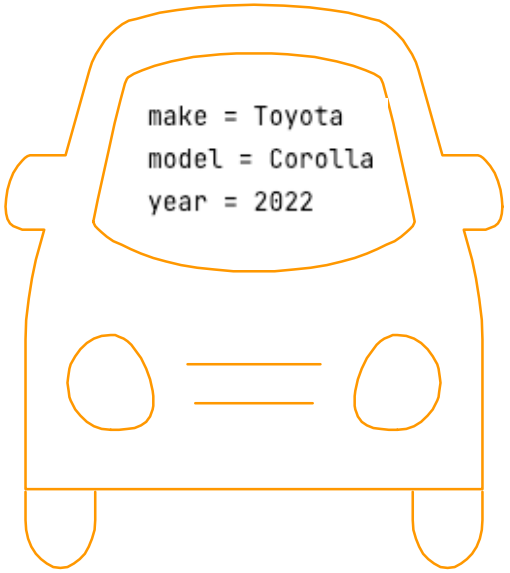
DEFAULT CONSTRUCTOR

- A constructor is called "**Default Constructor**" when it doesn't have any parameter.

If there is no constructor in a class, compiler automatically creates a default constructor

CONSTRUCTOR WITH NO AND ALL PARAMETERS

```
public class Car {
    private String make;
    private String model;
    private int year;

    public Car() { }

    public Car(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }
}
```



RULES FOR CREATING CONSTRUCTORS

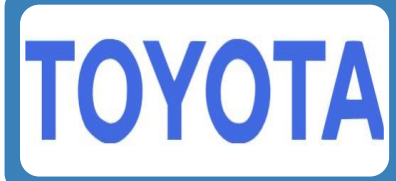
Constructor name must be the same as its class name

A Constructor must have no explicit return type

A Java constructor cannot be **abstract**, **static**, **final**, and **synchronized**

PARAMETERIZED CONSTRUCTORS

A constructor which has a specific number of parameters is called a parameterized constructor

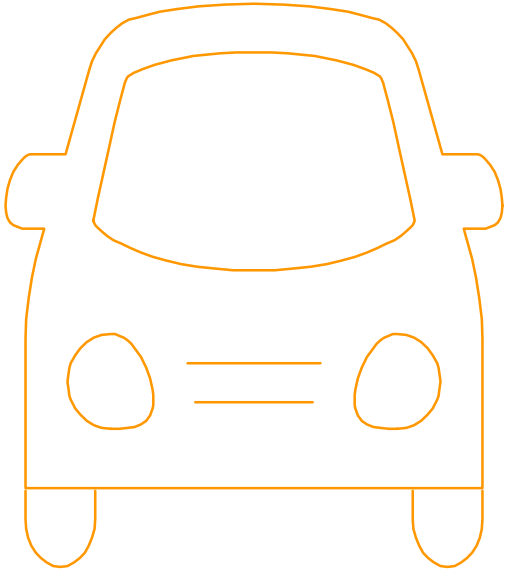


In the Car class, when you create a Toyota car object, you may want to only pass make or make and model or make, model and year

More than one constructor with different parameters can exist (called method overloading..)

CONSTRUCTOR WITH 1 AND 2 PARAMETERS

```
public Car(String make) {
    this.make = make;
}

public Car(String make, String model) {
    this.make = make;
    this.model = model;
}
```

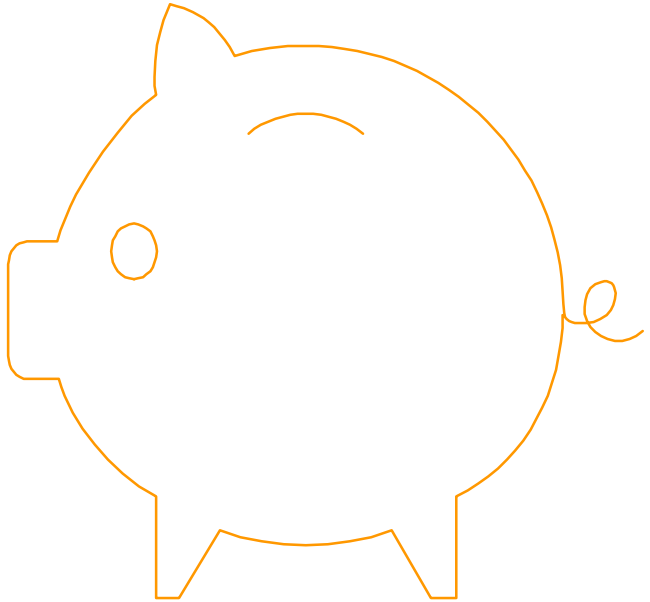


EXERCISE

Question:

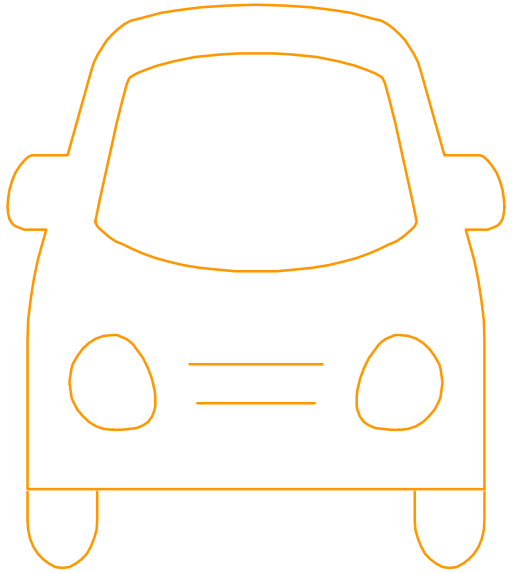
In the **Pig** class, add constructors with

- no argument,
- name only,
- name and color only,
- finally for name, color and age.



CREATING OBJECTS WITH PARAMETERS

```
public class CreateCars {
    public static void main(String[] args) {
        Car toyota = new Car();
        Car bmw = new Car("BMW");
        Car honda = new Car("Honda", "CRV");
        Car nissan = new Car("Nissan", "Sentra", 2010);
    }
}
```

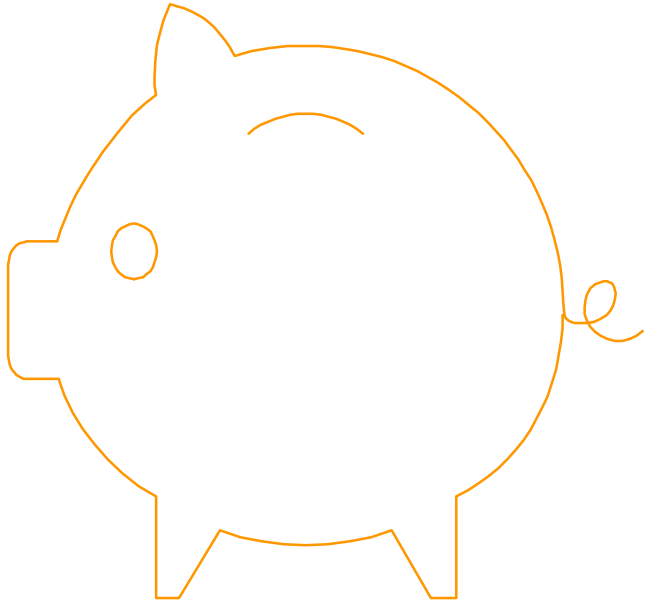


EXERCISE

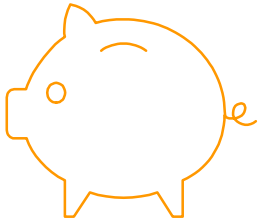
Question:

In the **CreatePigs** class, create several **pig** objects using constructors with

- no arguments
- name
- name and color
- name color and age



PIG – CONSTRUCTOR METHODS



```
public Pig() {
}

public Pig(String name) {
    this.name = name;
}

public Pig(String name, int age) {
    this.name = name;
    this.age = age;
}
```

```
public Pig(String name, int age, String color) {
    this.name = name;
    this.age = age;
    this.color = color;
}
```

toString() METHOD

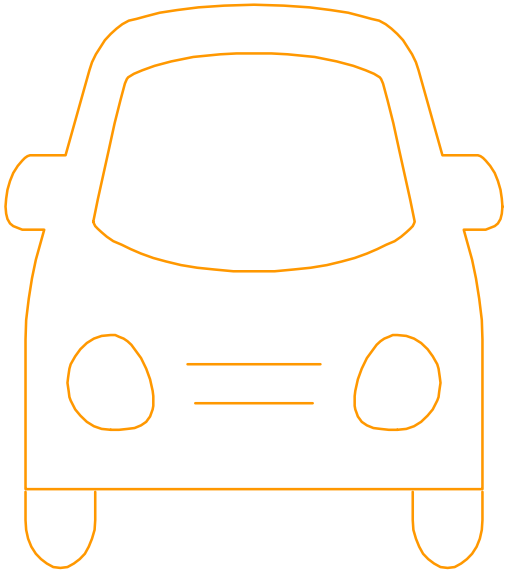
The **toString()** method returns the **String** representation of the object

If you want to print any object, Java compiler internally invokes the **toString()** method on the object

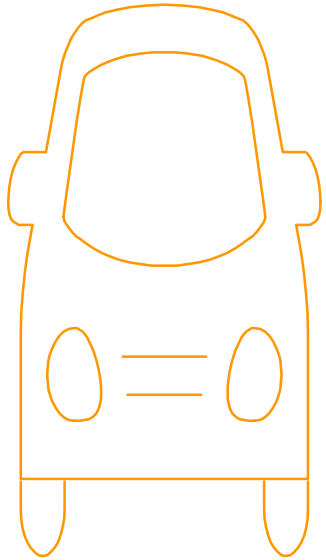
So, overriding the **toString()** method, returns the desired output, it can be the state of an object etc. depending on your implementation

OVERRIDDEN toString() METHOD

```
public class Car {

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



CREATING OBJECTS WITH PARAMETERS



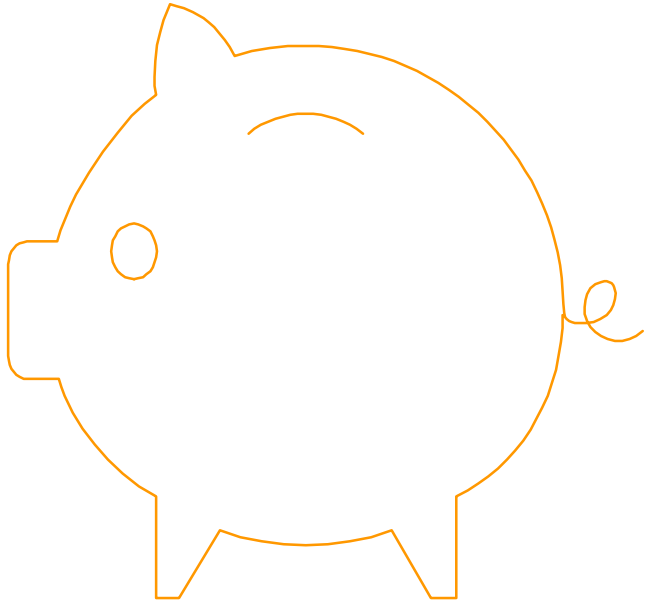
```
public class CreateCars {
    public static void main(String[] args) {
        Car toyota = new Car();
        Car bmw = new Car("BMW");
        Car honda = new Car("Honda", "CRV");
        Car nissan = new Car("Nissan", "Sentra", 2010);
        toyota.setMake("Toyota");toyota.setModel("Corolla");toyota.setYear(2022);
        System.out.println("toyota = " + toyota);
        System.out.println("nissan = " + nissan);
        System.out.println("honda = " + honda);
        System.out.println("bmw = " + bmw);
    }}
```

```
toyota = Car{make='Toyota', model='Corolla', year=2022}
nissan = Car{make='Nissan', model='Sentra', year=2010}
honda = Car{make='Honda', model='CRV', year=0}
bmw = Car{make='BMW', model='null', year=0}
```

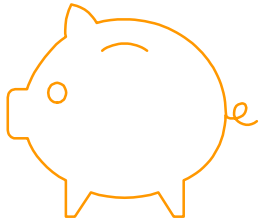
EXERCISE

Question:

In the **Pig** class, create **toString()** method which returns the class name along with name, age and color. Use **concat** operator “+” to construct the return message.



PIG : TO-STRING METHOD



```
@Override
public String toString() {
    return "Pig{" +
            "name='" + name + '\'' +
            ", age=" + age +
            ", color='" + color + '\'' +
            '}';
}
```

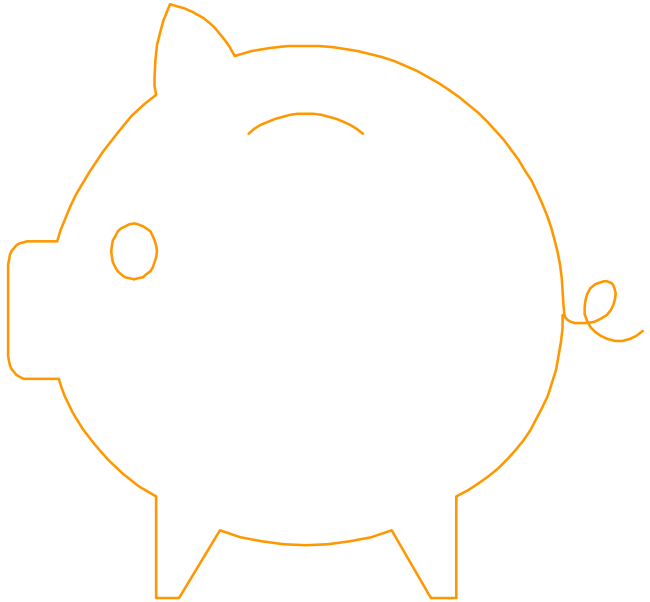
EXERCISE

Question:

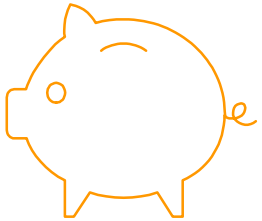
In the **CreatePigs** class, create **new Pigs** with different parameters.

- Pinky who is 9 years old
- Blacky whose color is black
- Dotty whose color is gray, and she is 2 years old

Print the values of the newly created pigs.



PIG – CREATE NEW PIGS AND PRINT



```
public class CreatePigs {
    public static void main(String[] args) {
        Pig rosyPig = new Pig();
        rosyPig.setName("Rosy");
        rosyPig.setColor("Red");
        rosyPig.setAge(4);
        System.out.println("name = " + rosyPig.getName());
        System.out.println("color = " + rosyPig.getColor());
        System.out.println("age = " + rosyPig.getAge());
        Pig pinky = new Pig("Pinky", 9);
        Pig blacky = new Pig("Blacky", "Black");
        Pig dotty = new Pig("Dotty",2, "Gray");
        System.out.println("pinky = " + pinky);
        System.out.println("blacky = " + blacky);
        System.out.println("dotty = " + dotty);
    }
}
```

EXERCISE – BOOK CLASS

Question:

Create **Book** Class with title (String), author(String), year (int), publisher(String), cost (double), isbn(String) attributes.

Create constructors with varying arguments and getters and setters along with toString() method.

EXERCISE – BOOK CLASS

Question:

Add a new class called CreateBooks and create 3 books.

Book1 = new Book()

Book2 = new Book("Java for QE")

Book3 = new Book("Java for Fed Reserve","Kangs")

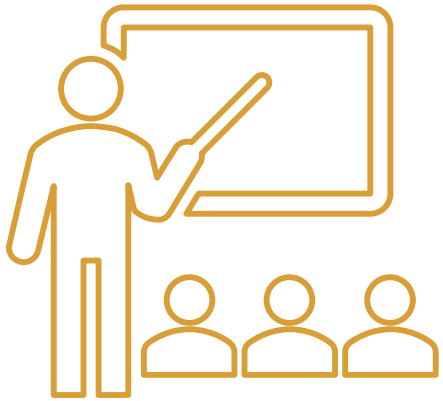
Print all the 3 books' information.

EXERCISE

Question:

Create **Course** Class with attributes:

name (String), credits (int), maxEnrollment (int), reservedSeat (int).



Create constructors with varying arguments and getters and setters along with toString() method.

EXERCISE

Question:

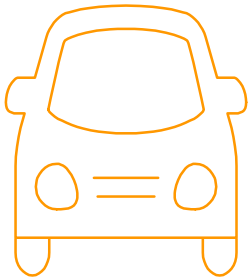
Create **Student** Class with attributes:

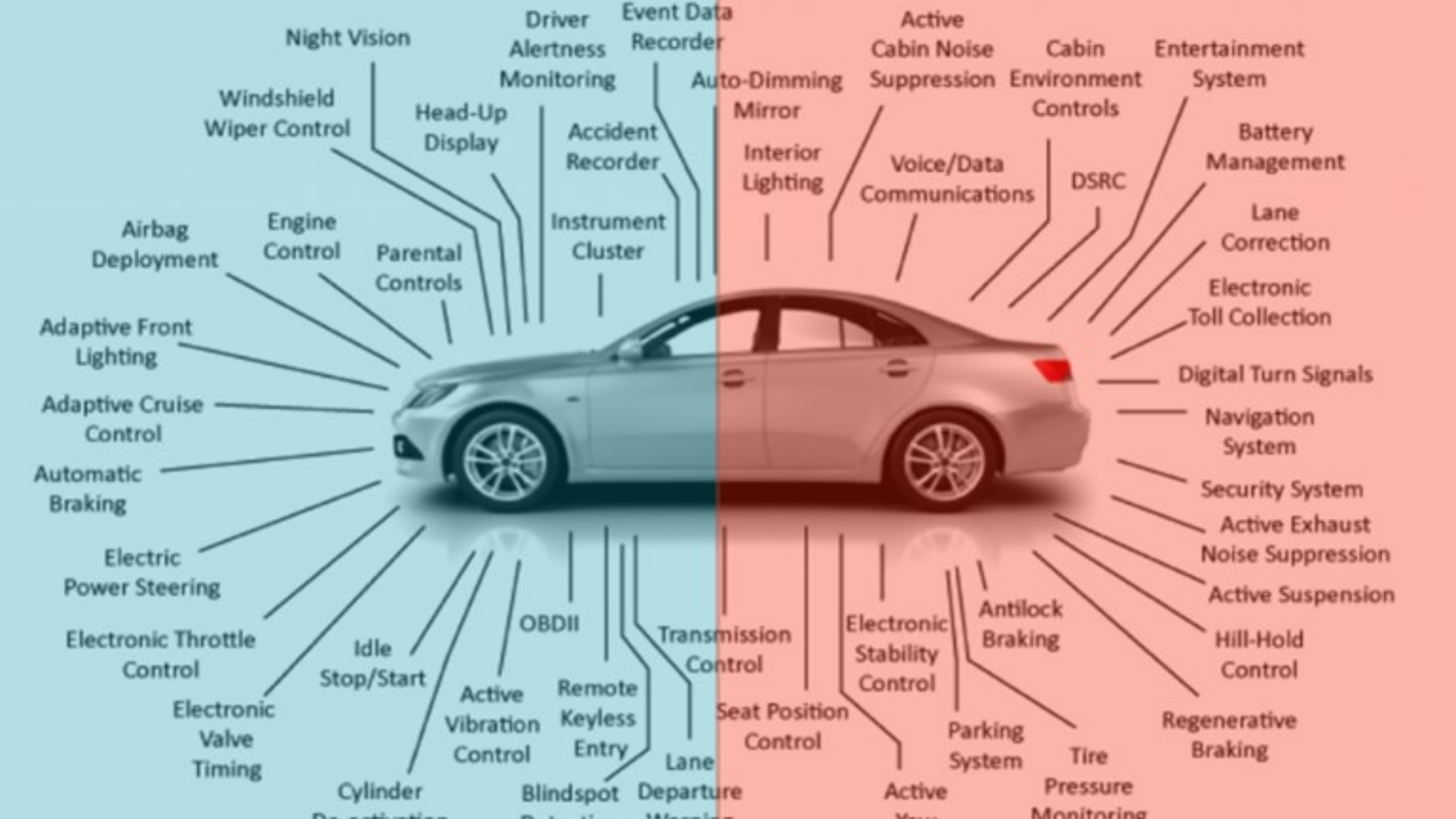
lastName, firstName, id (int),entranceYear (int), facultyType (String), totalYears(int), gpa (double).

Create constructors with varying arguments and getters and setters along with toString() method.

OBJECT DECOMPOSITION

DECOMPOSE INTO SMALLER OBJECTS




Night Vision
Driver Alertness Monitoring
Event Data Recorder
Active Cabin Noise Suppression
Cabin Environment Controls
Entertainment System
Windshield Wiper Control
Head-Up Display
Auto-Dimming Mirror
Accident Recorder
Interior Lighting
Voice/Data Communications
DSRC
Battery Management
Airbag Deployment
Engine Control
Parental Controls
Instrument Cluster
Lane Correction
Electronic Toll Collection
Adaptive Front Lighting
Digital Turn Signals
Adaptive Cruise Control
Navigation System
Automatic Braking
Security System
Active Exhaust Noise Suppression
Electric Power Steering
Active Suspension
Electronic Throttle Control
OBDII
Transmission Control
Electronic Stability Control
Antilock Braking
Hill-Hold Control
Idle Stop/Start
Active Vibration Control
Remote Keyless Entry
Seat Position Control
Parking System
Regenerative Braking
Electronic Valve Timing
Lane Departure
Tire Pressure Monitoring
Cylinder
Blindspot
Active

WHAT IS OBJECT DECOMPOSITION?

- Dividing a large complex system into a hierarchy of smaller components with lesser complexities

Each major component of the system is called a subsystem

ADVANTAGES OF DECOMPOSITION

- The individual components are of lesser complexity, and so more understandable and manageable.
- It enables division of workforce having specialized skills.
- It allows subsystems to be replaced or modified without affecting other subsystems


Thank You!

THANKS!

Your feedback is welcome
support@kavinschool.com