# Objects, Classes, Interfaces, Packages, and Inheritance

Introduction to basic OOP concepts in Java:

- Objects
- Classes
- Inheritance
- Interfaces
- Packages

Each concept relates to the real world while introducing Java syntax.

# What is an Object?

A software bundle of related state and behavior.

### Key characteristics:

- State (represented by fields/variables)
- Behavior (represented by methods/functions)

### Real-world examples:

- Dogs: state (name, color), behavior (barking)
- Bicycles: state (speed, gear), behavior (changing gear)

### Object Benefits

Key advantages of objects:

1. Modularity: Independent code units
2. Information-hiding: Internal details hidden
3. Code re-use: Existing objects can be reused
4. Pluggability: Easy to replace objects
5. Debugging ease: Fix/replace individual objects

## Classes:

A class is the blueprint from which individual objects are created.

Example Bicycle class defines:

- Fields (state): cadence,speed,and gear.
- Methods (behavior): changeCadence, changeGear, speedUp, applyBrakes, and printStates.

```
class Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence(int newValue) {
          cadence = newValue;
    }

    void changeGear(int newValue) {
         gear = newValue;
    }

    void speedUp(int increment) {
         speed = speed + increment;
    }

    void applyBrakes(int decrement) {
         speed = speed - decrement;
    }

    void printStates() {
         System.out.println("cadence:" +
             cadence + " speed:" +
             speed + " gear:" + gear);
    }
}
```

## Class Usage Example

Creating and using Bicycle objects:

```
class BicycleDemo {
    public static void main(String[] args) {

        // Create two different
        // Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on
        // those objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike1.printStates();

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
        bike2.printStates();
    }
}
```

```
[finished]

cadence:50 speed:10 gear:2
cadence:40 speed:20 gear:3
```

## Inheritance

Different kinds of objects often share common characteristics. Example: Mountain bikes, road bikes, and tandem bikes all have:

- Current speed
- Pedal cadence
- Current gear

Each type adds unique features:

### ▒▒▒ Tandem bikes: Two seats & handlebars

### ▒▒▒ Road bikes: Drop handlebars


### ▒▒▒ Mountain bikes: Additional chain ring

# Inheritance

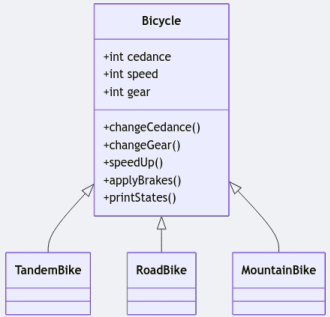
## ▓▓ Inheritance allows classes to inherit state and behavior from other classes.

Key concepts:

- **Superclass** (Parent class, e.g., Bicycle)
- **Subclass** (Child class, e.g., MountainBike)
- Java allows single inheritance (one direct superclass)
- Unlimited subclasses per superclass

```
class TandemBike extends Bicycle {
    // inherits all fields and methods from Bicycle
}
```

## ▓▓ Class Diagram Hierarchy Example

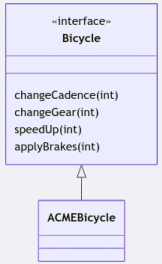


# Interfaces

## In its most common form, an interface is a group of related methods with empty bodies.

```
interface Bicycle {
    // wheel revolutions per minute
    void changeCadence(int newValue);
    void changeGear(int newValue);
    void speedUp(int increment);
    void applyBrakes(int decrement);
}
```

## Interface Class Diagram




## Implementing an Interface

1. Change the class name (e.g., to ACMEBicycle)
2. Use the implements keyword

```
class ACMEBicycle implements Bicycle {
    int cadence = 0;
    int speed = 0;
    int gear = 1;

    // Interface methods must be implemented
    void changeCadence(int newValue) {
        cadence = newValue;
    }
    // implements all other interface methods ...
    //
}
```

## A **package** is a namespace that organizes related classes and interfaces.

Think of it like folders on your computer:

- HTML files in one folder
- Images in another
- Scripts in a separate folder

Java programs can have hundreds of classes—packages keep them organized.

## Java's Built-in Packages

The Java platform provides a vast **class library** (set of packages) called the **API** (Application Programming Interface).

These packages handle common tasks, such as:

- `String`: Text manipulation
- `File`: Create, delete, or modify files
- `Socket`: Network communication
- **GUI components**: Buttons, checkboxes, etc.

Thousands of prebuilt classes let you focus on your app's logic.

## The Java API Documentation

The **Java Platform API Specification** lists all packages, classes, and methods in Java SE.

## References

https://dev.java/learn/oop/