Object-oriented Programming

- Object-oriented programming:
  - Group data and procedures into objects ↔
  - Models of state and behaviour of real world objects
  - state „fields“ ; behaviour „methods“

- Methods should mainly act on an object’s fields
- **Classes**: Blueprints for objects → **Objects**: Instances of classes
- **Advantages**
  - Intuitive models
  - Information hiding
  - Increased modularity, locality etc.
  - Increased code re-use
  - etc.

---

```java
class BicycleDemo {
    public static void main(String[] args) {
        // Create two different Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

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

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

---

Inheritance

Inheritance: Define new, more specialized classes from existing classes

```java
class RoadBike extends Bicycle {
    // additional fields and methods
    // that define a road bike
    // go here
}
```
**Inheritance**

- Define new, more *specialized* classes from existing classes.

```java
class RoadBike extends Bicycle {
    // additional fields and methods
    // that define a road bike
    // go here
}
```

---

**Interfaces**

- Specify in an abstract way what a class implementing that interface should exhibit as behaviours (create blueprint for blueprints).

```java
interface IBicycle {
    void changeCadence(int newValue);
    void changeGear(int newValue);
    void speedUp(int increment);
    void applyBrakes(int decrement);
}
```

```java
class Bicycle implements IBicycle {
    // remainder of this class implemented as before
    // except that above methods must be public
}
```

---

**Example:**

```java
class Manual {
    height: int
    weight: int
    eatSomething() desirable;
    implement
}
```

```java
class ICantDive {
    height: int
    weight: int
    eatSomething() desirable;
    implement
}
```

```java
class Vehicle {
    height: int
    weight: int
    changeDive() desirable;
    accelerate() desirable;
    brake() desirable;
    implement
}
```

```java
class Whale {
    height: int
    weight: int
    changeDive() desirable;
    accelerate() desirable;
    dive() desirable;
    implement
}
```

```java
class Submarine {
    height: int
    weight: int
    changeDive() desirable;
    accelerate() desirable;
    dive() desirable;
    implement
}
```