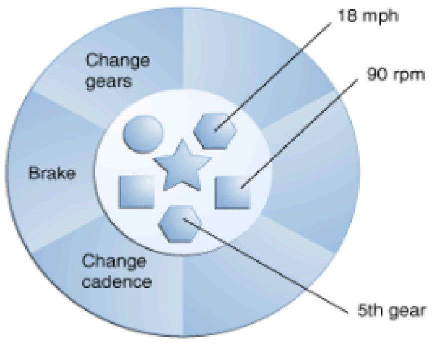
Object Oriented Programming Concepts

Object Oriented Programming Concepts
- What Is an Object?
- What is Class ?
- What is Inheritance ?
- What Is a Package ?
- What is An Interface ?

What Is an Object?

- Software objects are conceptually similar to real-world objects: they too consist of state and related behavior.
- An object stores its state in *fields* and exposes its behavior through *methods*.
- Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication
- Hiding internal state and requiring all interaction to be performed through an object's methods is known as ***data encapsulation* — a fundamental principle of object-oriented programming.**



A bicycle modeled as a software object.

Bundling code into individual software objects provides a number of benefits, including:

1. **Modularity:** The source code for an object can be written and maintained independently of the source code for other objects.
2. **Information-hiding:** By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
3. **Code reuse:** If an object already exists, you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects.
4. **Pluggability and debugging ease:** If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace *it*, not the entire machine.

What is Class ?

- In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles.
- A *class* is the blueprint from which individual objects are created.

The following `Bicycle` class is one possible implementation of a bicycle:

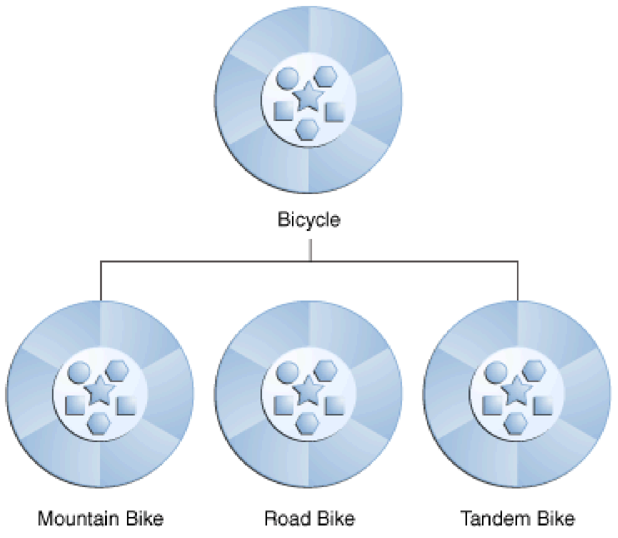
```
class Bicycle {

    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);
    }
}
```

Here's a `BicycleDemo` class that creates two separate `Bicycle` objects and invokes their methods:

```
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();
    }
}
```

What is Inheritance ?

- Different kinds of objects often have a certain amount in common with each other.
- Mountain bikes, road bikes, and tandem bikes, for example, all share the characteristics of bicycles (current speed, current pedal cadence, current gear).
- Yet each also defines additional features that make them different: tandem bicycles have two seats and two sets of handlebars;
- **Object-oriented programming allows classes to *inherit* commonly used state and behavior from other classes.**
- In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of *subclasses*:



A hierarchy of bicycle classes.

What Is a Package ?

- A package is a namespace that organizes a set of related classes and interfaces.
- Because software written in the Java programming language can be composed of hundreds or *thousands* of individual classes, it makes sense to keep things organized by placing related classes and interfaces into packages.
- The Java Platform API Specification contains the complete listing for all packages, interfaces, classes, fields, and methods supplied by the Java SE platform.

What is An Interface ?

- In its most common form, an interface is a group of related methods with empty bodies.
- A bicycle's behavior, if specified as an interface, might appear as follows:

```
interface Bicycle {

    //  wheel revolutions per minute
    void changeCadence(int newValue);

    void changeGear(int newValue);

    void speedUp(int increment);

    void applyBrakes(int decrement);
}
```

- To implement this interface, the name of your class would change (to a particular brand of bicycle, for example, such as `ACMEBicycle`), and you'd use the `implements` keyword in the class declaration:

```
class ACMEBicycle implements Bicycle {

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

   // The compiler will now require that methods
   // changeCadence, changeGear, speedUp, and applyBrakes
   // all be implemented. Compilation will fail if those
   // methods are missing from this class.

    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);
    }
}
```

- Implementing an interface allows a class to become more formal about the behavior it promises to provide.
- Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler.
- If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.