

Fundamentals of Programming

Laboratory 8

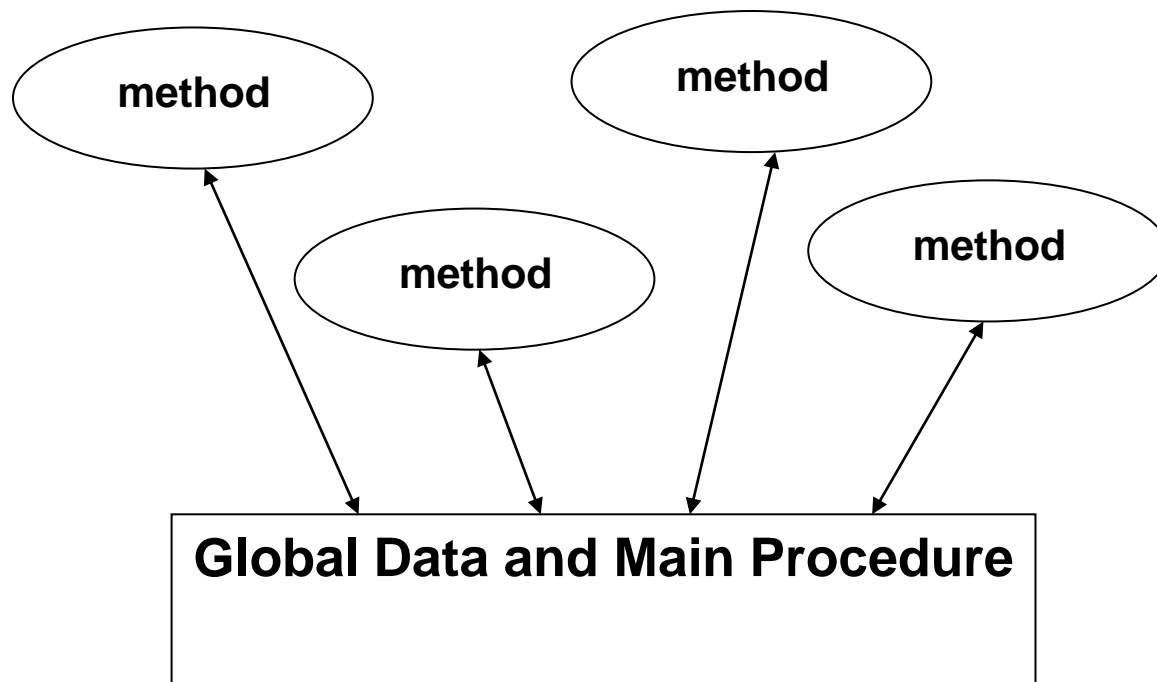
Classes and Objects

Object Oriented Programming



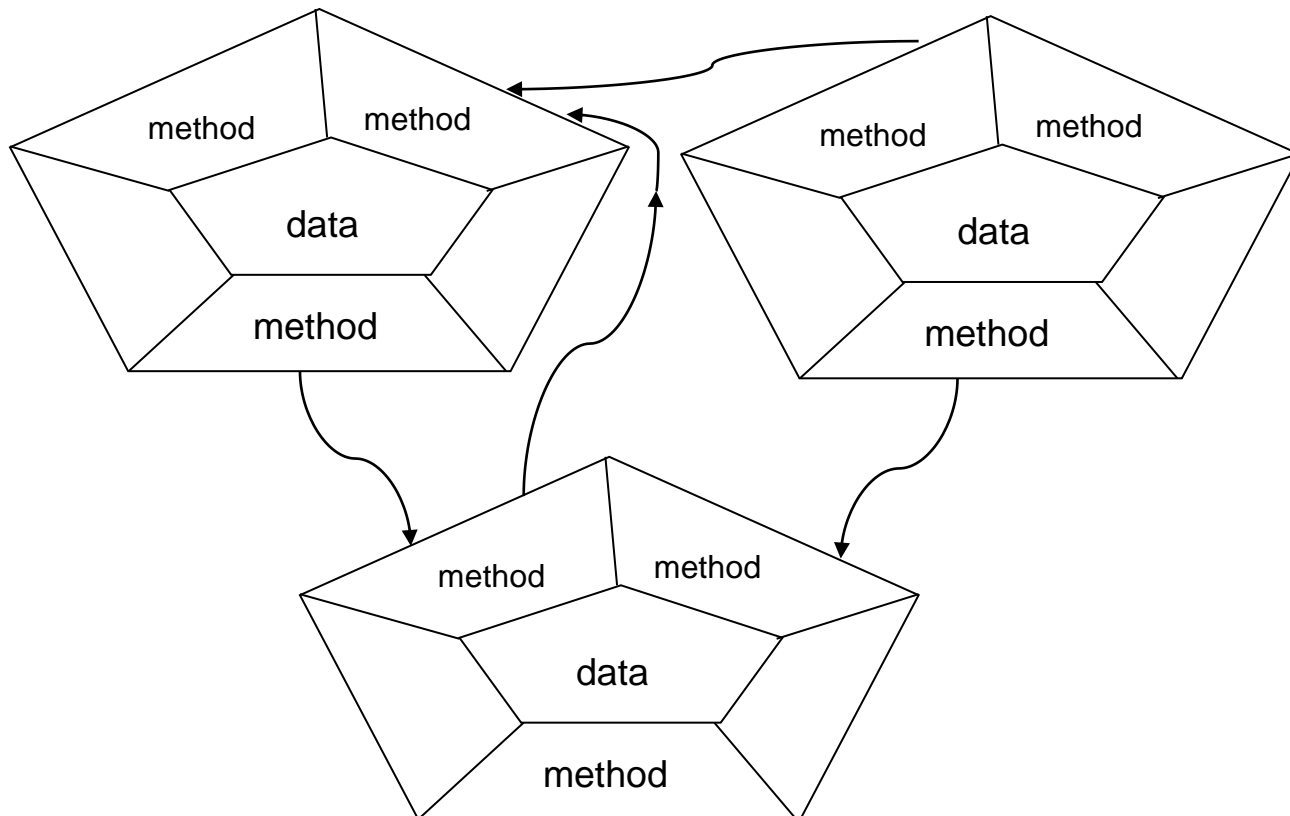
Procedural Programming

- variables are operated on by subprograms
- separation of data and functionality
- difficult to modify and expand



Object-Oriented Programming

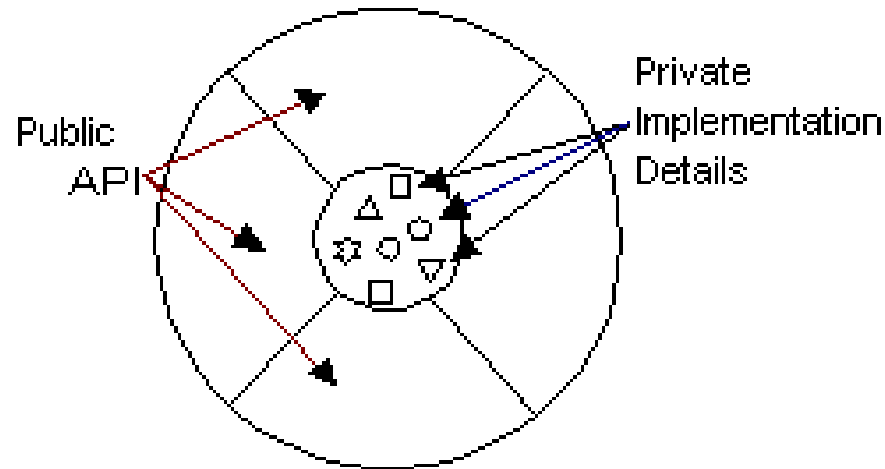
- objects serve as a combination of data (attributes, state) and functionality (behaviors)
- integration of data and functionality



CLASS

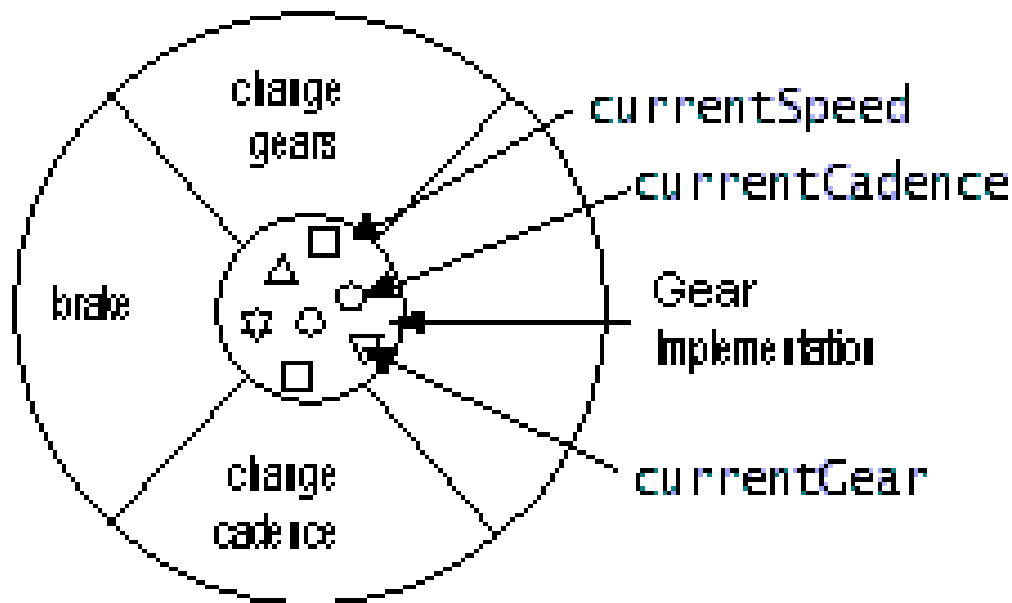
A class is a prototype that defines the variables and the methods common to all objects of a certain kind.

Visual representation of class:



BIKE CLASS

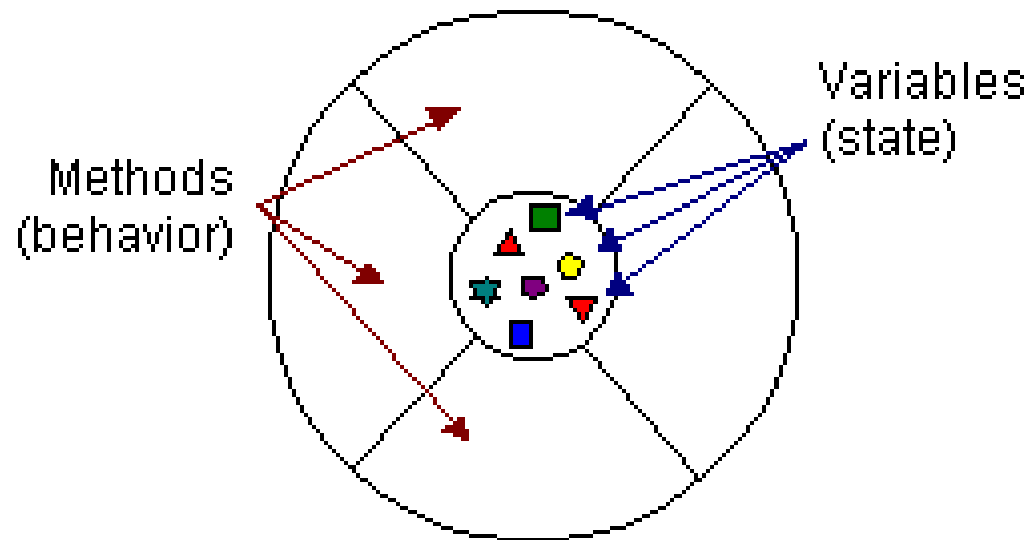
Visual representation of a bike class:



OBJECT

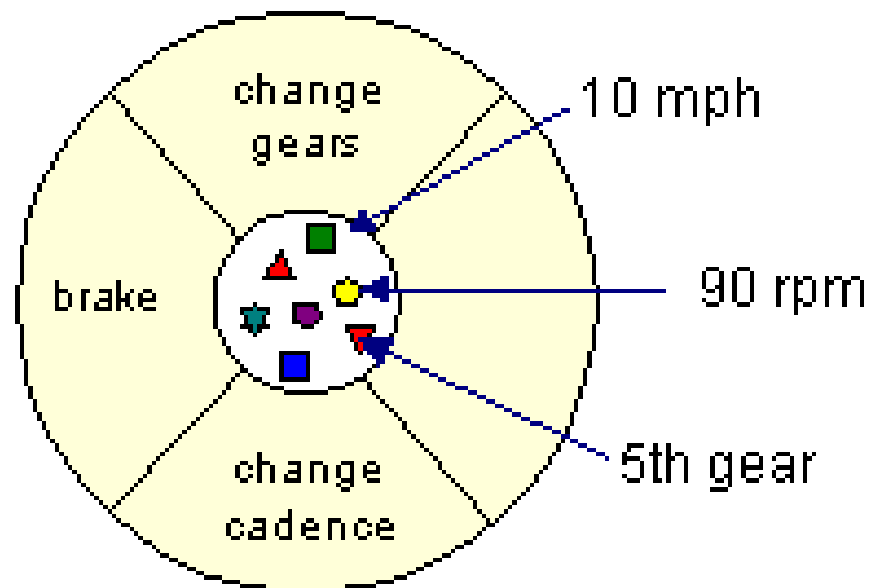
An object is a software bundle of variables and related methods.

Visual representation of a software object:



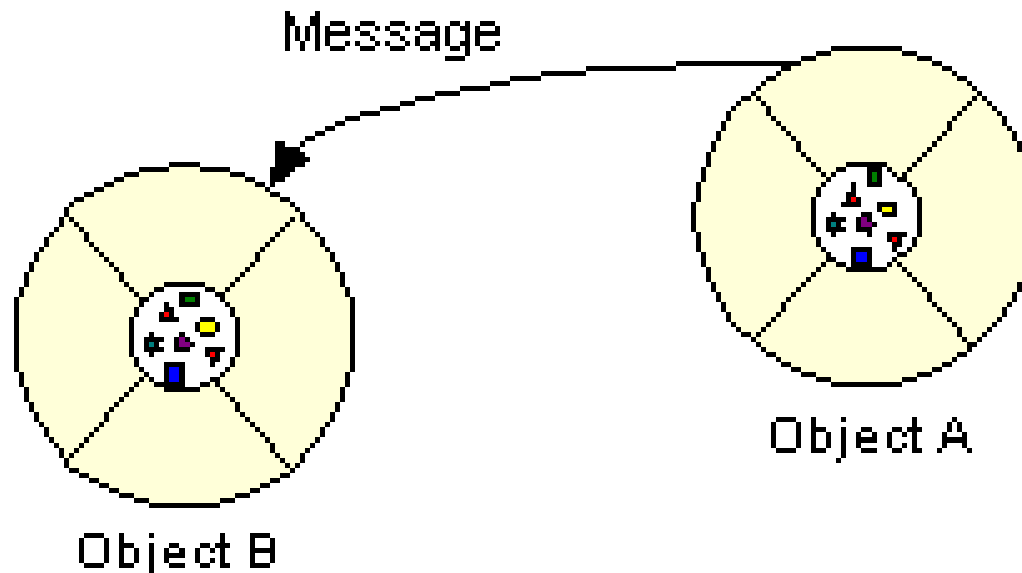
BIKE OBJECT

Bicycle modeled as a software object:



Message

Software objects interact and communicate with each other by sending ***messages*** to each other. When object A wants object B to perform one of B's methods, object A sends a message to object B

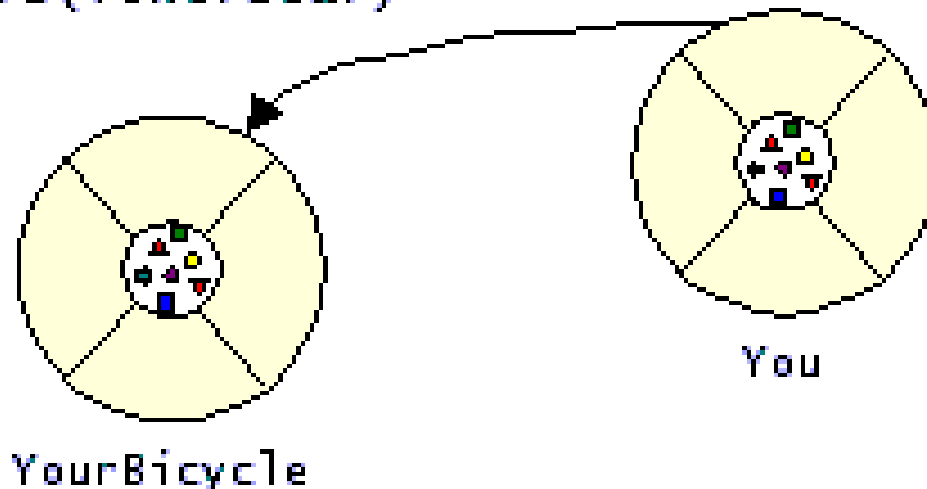


Message

The three components of a message:

- 1) The object to which the message is addressed (YourBicycle)
- 2) The name of the method to perform (changeGears)
- 3) Any parameters needed by the method (lowerGear)

`changeGears(lowerGear)`



Questions

1. What are the two things that make up an object?

- attributes and behavior
- commands and data files
- spit and vinegar



ORAL
EXERCISE

Questions

2. In a Java class, a method is an example of what?

- Attributes
- Statements
- Behavior



Classes and Objects

The classes you have already used are **Scanner**, **System** and **String**.

When we use a class, we can construct **objects** which behave in the way defined by that class. You have already constructed objects in Java:

```
Scanner input = new Scanner(System.in) ;
```

This line of code constructs a new **Scanner object**, which we have called **input**. The parameter **System.in** tells the new object to scan input from the keyboard.

Classes and Objects

In most of our programs we named the Scanner object **input**.

We use a dot (.) in between the object name and the method whenever we call it :

```
input.nextLine ( );
```

```
input.nextInt ( );
```

```
input.nextDouble ( );
```

Classes and Objects

The **Scanner class** has been written by the developers of Java. The diagram shows a simplified representation of the class:

Scanner	The class name
value	The value scanned
next nextInt nextDouble	Things we can do with Scanner objects

Declaring Classes

```
class MyClass
{
    //field, constructor, and method
    declarations
}
```

The *class body* contains all the code that provides for the life cycle of the objects created from the class: constructors for initializing new objects, declarations for the fields that provide the state of the class and its objects, and methods to implement the behavior of the class and its objects

Object Variables

When a number of objects are created from the same class blueprint, they each have their own distinct copies of *instance variables*.

In the case of the Bicycle class, the instance variables are cadence, gear, and speed.

Each Bicycle object has its own values for these variables, stored in different memory locations

Class Variables

Sometimes, you want to have variables that are common to all objects. This is accomplished with the ***static*** modifier.

Fields that have the static modifier in their declaration are called *static fields* or *class variables*. They are associated with the class, rather than with any object.

Class Variables

```
public class Bicycle
{ private int cadence;
  private int gear;
  private int speed;
  // add an instance variable for the object ID
  private int id;
  // add a class variable for the number of Bicycle
  objects instantiated
  private static int numberOfBicycles = 0;
  .....
}
```

Questions

3. If you want to make a variable a class variable, what statement must you use when it is created?

- new
- public
- static



Creating Objects

Class provides the blueprint for objects; you create an object from a class. Each of the following statements creates an object and assigns it to a variable:

```
Point originOne = new Point(23, 94);  
Rectangle rectOne = new Rectangle(originOne,  
    100, 200);  
Rectangle rectTwo = new Rectangle(50, 100);
```

The first line creates an object of the *Point* class, and the second and third lines each create an object of the *Rectangle* class.

Using Objects

Object fields are accessed by their name. You must use a name that is unambiguous. You may use a simple name for a field within its own class.

For example, we can add a statement *within* the Rectangle class that prints the width and height:

```
System.out.println("Width and height are: " +  
width + ", " + height);
```

In this case, width and height are simple names.

Using Objects

Code that is outside the object's class must use an object reference or expression, followed by the dot (.) operator, followed by a simple field name, as in:

objectReference.fieldName



Using Objects

To access a field, you can use a named reference to an object or you can use any expression that returns an object reference.

Recall that the new operator returns a reference to an object.

```
int height = new Rectangle().height;
```

This statement creates a new Rectangle object and immediately gets its height. In essence, the statement calculates the default height of a Rectangle.



Calling an Object's Methods

You also use an object reference to invoke an object's method.

```
objectReference.methodName (argumentList) ;
```

```
//if no arguments
```

```
objectReference.methodName () ;
```


Calling an Object's Methods

Example: The Rectangle class has two methods:
 getArea() to compute the rectangle's area and
 move() to change the rectangle's origin.

```
System.out.println("Area of rectOne: " +  
    rectOne.getArea());
```

```
...
```

```
rectTwo.move(40, 72);
```

Constructors

A class contains **constructors** that are invoked to create objects from the class blueprint.

Constructor declarations look like method declarations - except that they use the name of the class and have no return type.

Constructors

For example, Bicycle has one constructor:

```
public Bicycle(int startCadence, int  
    startSpeed, int startGear)  
{  
    gear = startGear;  
    cadence = startCadence;  
    speed = startSpeed;  
}
```

```
Bicycle myBike = new Bicycle(30, 0, 8);
```

To create a new Bicycle object called myBike, a constructor is called by the new operator.
new Bicycle(30, 0, 8) creates space in memory for the object and initializes its fields.

Constructors

Although Bicycle only has one constructor, it could have others, including a no-argument constructor:

```
public Bicycle()  
{ gear = 1;  
  cadence = 10;  
  speed = 0;  
}
```

```
Bicycle yourBike = new Bicycle();
```

invokes the no-argument constructor to create a new Bicycle object called yourBike.

Constructors

The Java platform differentiates constructors on the basis of the number of arguments in the list and their types.

You don't have to provide any constructors for your class, but you must be careful when doing this. The compiler automatically provides a no-argument, default constructor for any class without constructors.

Questions

4. Consider the following class:

```
public class IdentifyMyParts
{
    public static int x = 7;
    public int y = 3;
}
```

- a. What are the class variables?
- b. What are the instance variables?

Answer 1a: x, 1b: y



BOARD
EXERCISE

Questions

5. What is the output from the following code?

```
IdentifyMyParts a = new IdentifyMyParts();  
IdentifyMyParts b = new IdentifyMyParts();  
a.y = 5;  
b.y = 6;  
a.x = 1;  
b.x = 2;  
System.out.println("a.y = " + a.y);  
System.out.println("b.y = " + b.y);  
System.out.println("a.x = " + a.x);  
System.out.println("b.x = " + b.x);  
System.out.println("IdentifyMyParts.x = " +  
    IdentifyMyParts.x);
```



BOARD
EXERCISE

Questions

6. What's wrong with the following program?

```
public class SomethingIsWrong {  
    public static void main(String[] args) {  
        Rectangle myRect;  
        myRect.width = 40;  
        myRect.height = 50;  
        System.out.println("myRect's area is  
" + myRect.area() );  
    }  
}
```



Task 1:

Write a class Person with FirstName and LastName as data members.

Write the myNameIs() method to return the full name of the person.

Define constructors to take appropriate parameters.

```
public static void main (String [] args)
{
    Person teacher = new Person("Michal", "Bujacz");
    System.out.print(teacher.myNameIs());
}
```



```
class Person
```

```
{  
    public String lastName;  
    public String firstName;  
  
    public String myNameIs ()  
    {  
        return ( firstName + " " + lastName);  
    }  
    public Person( String a, String b)  
    {  
        firstName = a;  
        lastName = b;  
    }  
}
```



Task 2:

Given the following class, called NumberHolder, write some code that creates an instance of the class, initializes its two member variables, and then displays the value of each member variable.

```
public class NumberHolder
{
    public int anInt;
    public float aFloat;
}
```



Task 3:

Write a Java application that takes an argument as a string, converts it to a float variable, converts that to a Float object, and finally turns that into an int variable. Run it a few times with different arguments to see what results.



Task 4:

Write a main method which creates a new Property object with id 0, price 550000, area 2100 and availability true.

```
public static void main (String [] args)
{
    Property house1 = new Property(0,550000,2100,true);
    house1.printPropertyAttributes();
    System.out.print("Is the house available? " +
        house1.isAvailable());
}
```

