Lecture 5: Introduction to Classes and Objects

Object Oriented Concepts and Programming (CSC244)

By

Dr. Tabbasum Naz

tabbasum.naz@ciitlahore.edu.pk

Object

- Objects are key to understanding *object-oriented* technology.
- Examples of real-world objects: your dog, your desk, your television set, your bicycle.
- Real-world objects share two characteristics: They all have ***state*** and ***behavior***.
  - **Dogs** have **state (name, color, breed, hungry)** and **behavior (barking, fetching, wagging tail)**.
  - **Bicycles** also have **state (current gear, current pedal cadence, current speed)** and **behavior (changing gear, changing pedal cadence, applying brakes).**
- Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

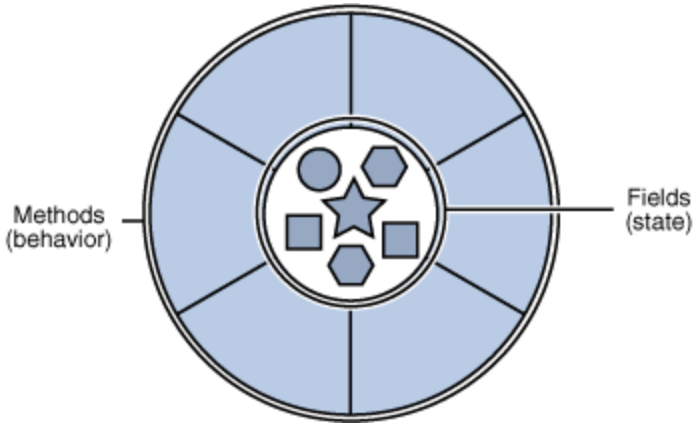
Objects (Contd)

- Take a minute right now to observe the real-world objects that are in your immediate area.
- For each object that you see, ask yourself two questions:
  - "What possible states can this object be in?"
  - "What possible behavior can this object perform?“
- Real-world objects vary in complexity;
  - your **desktop lamp** may have only two possible **states (on and off)** and two possible **behaviors (turn on, turn off),**
  - **Desktop radio** might have additional **states (on, off, current volume, current station)** and **behavior (turn on, turn off, increase volume, decrease volume, seek, scan, and tune).**

Software Object

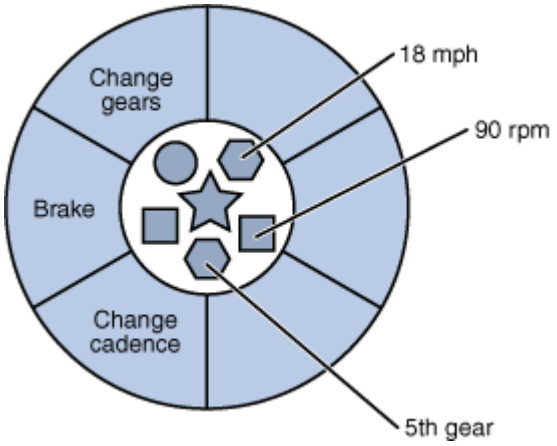
- Software objects are conceptually similar to real-world objects:
- They too consist of state and related behavior.
- An object stores its state in *fields* (variables in some programming languages) and exposes its behavior through *methods* (functions in some programming languages).
- Methods operate 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.

Software Object (Contd)



Bicycle Software Object

- By attributing state (current speed, current pedal cadence, and current gear) and providing methods for changing that state, the object remains in control of how the outside world is allowed to use it.
- For example, if the bicycle only has 6 gears, a method to change gears could reject any value that is less than 1 or greater than 6.



Class

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

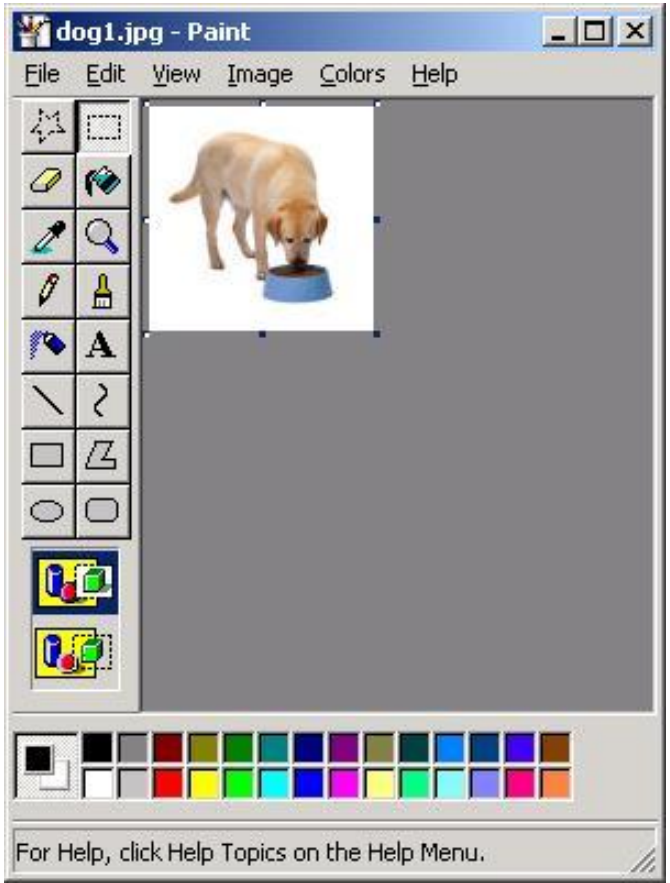
More on Classes and Objects

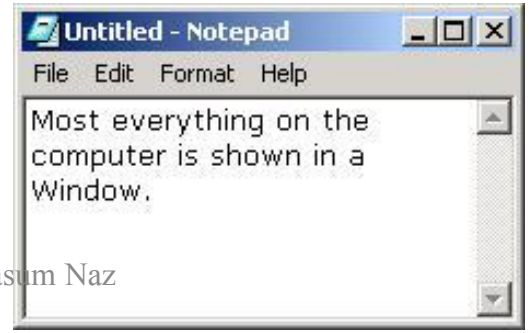
- A Java program consists of one or more classes
- A class is an abstract description of objects
- Class is a template for an object
- Class defines a new data type
- Once defined, this new data type can be used to create objects of that type
- An object is instance of a class.
- Because an object is an instance of a class, you will often see the two words object and instance interchangeably
- Here is an example class:
  - class Dog { *...description of a dog goes here...* }
- Here are some objects of that class:




More Objects

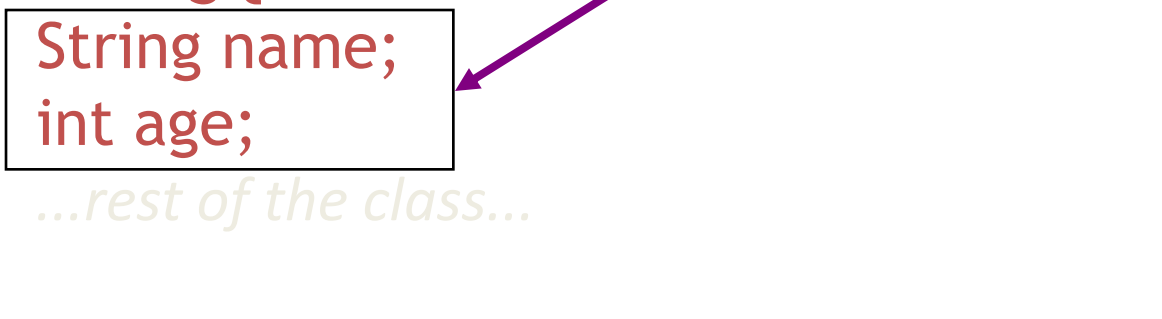
- Here is another example of a class:
  - class Window { … }
- Here are some examples of Windows:





Classes contain data definitions

- Classes describe the data held by each of its objects
- Example:
  - class Dog {

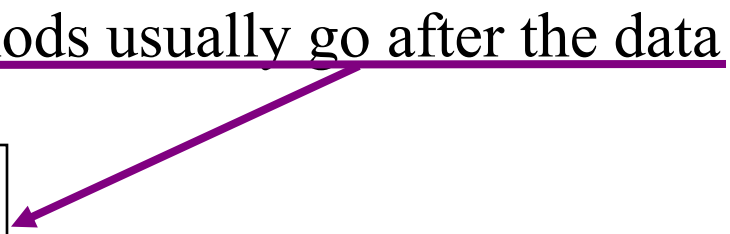


  }
- A class may describe any number of objects
  - Examples: "Fido", 3; "Rover", 5; "Spot", 3;

Classes contain methods

- A class may contain methods that describe the behavior of objects
- Example:
  - class Dog {

```
...
void bark() {
    System.out.println("Woof!");
}
}
```



- When we ask a *particular* Dog to bark, it says “Woof!”
- Only Dog *objects* can bark; the *class* Dog cannot bark

Methods contain statements

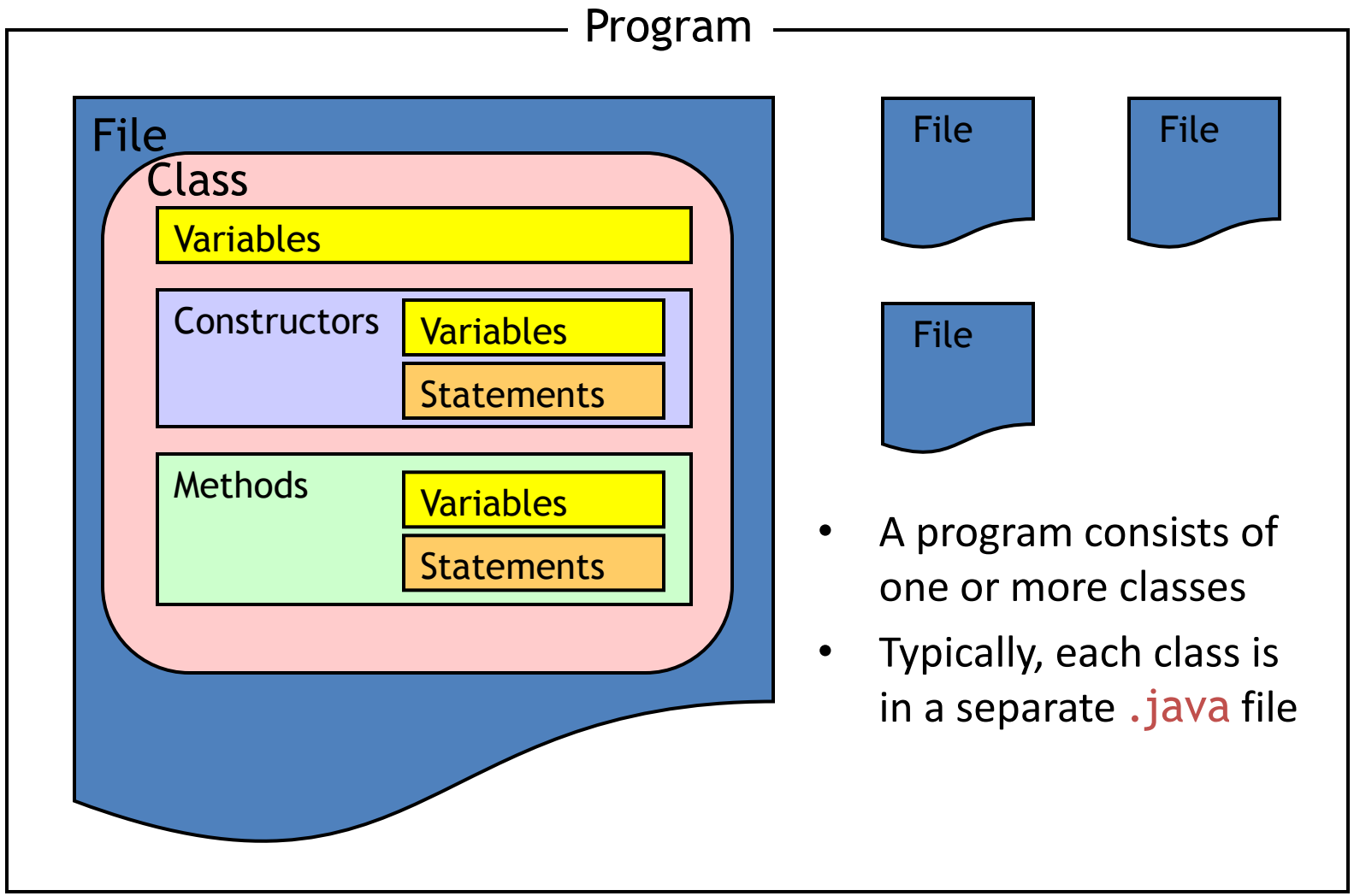
- A statement causes the object to do something
  - (A better word would be “command”—but it isn’t)
- Example:
  - System.out.println("Woof!");
  - This causes the particular Dog to “print” (actually, display on the screen) the characters Woof!

Classes always contain constructors

- A constructor is a piece of code that "constructs," or creates, a new object of that class
- If you don't write a constructor, Java defines one for you (behind the scenes)
- You can write your own constructors
- Example:
  - class Dog {
    String name;
    int age;
    Dog(String n, int age) {
    name = n;
    this.age = age;
    }
    }

(This part is the constructor)

Diagram of program structure



- A program consists of one or more classes
- Typically, each class is in a separate `.java` file

Bicycle Class

```
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);
    }
}
```

Fields

Methods

Bicycle class description

- You may have noticed that the Bicycle class does not contain a main method. That's because it's not a complete application; it's just the blueprint for bicycles that might be *used* in an application. The responsibility of creating and using new Bicycle objects belongs to some other class in your application.

BicycleDemo

```
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();
    }
}
```

Output

cadence:50 speed:10 gear:2

cadence:40 speed:20 gear:3

Box Class

```
// This program includes a method inside the box class.

class Box {
  double width;
  double height;
  double depth;

  // display volume of a box
  void volume() {
    System.out.print("Volume is ");
    System.out.println(width * height * depth);

  }
}
```

```
class BoxDemo3 {
  public static void main(String args[]) {
    Box mybox1 = new Box();
    Box mybox2 = new Box();

    // assign values to mybox1's instance variables
    mybox1.width = 10;
    mybox1.height = 20;
    mybox1.depth = 15;

    /* assign different values to mybox2's
       instance variables */
    mybox2.width = 3;
    mybox2.height = 6;
    mybox2.depth = 9;

    // display volume of first box
    mybox1.volume();

    // display volume of second box
    mybox2.volume();
  }
}
```

Output

```
Volume is 3000.0
Volume is 162.0
```

Box Class

```
/* A program that uses the Box class.

   Call this file BoxDemo.java
*/
class Box {
  double width;
  double height;
  double depth;
}
```

```
// This class declares an object of type Box.
class BoxDemo {
  public static void main(String args[]) {
    Box mybox = new Box();
    double vol;

    // assign values to mybox's instance variables
    mybox.width = 10;

    mybox.height = 20;
    mybox.depth = 15;

    // compute volume of box
    vol = mybox.width * mybox.height * mybox.depth;
        System.out.println("Volume is " + vol);
      }
    }
```

A complete “Dog” program

```
class Dog {
    String name;
    int age;

    Dog(String n, int age) {
        name = n;
        this.age = age;
    }

    void bark() {
      System.out.println("Woof!");
    }

    public static void main(String[ ] args) {
        Dog fido = new Dog("Fido", 5);
        fido.bark();
    }


} // ends the class
```

Another “Dog” program

```
class Dog {
    String name;
    int age;

    Dog(String n, int age) {
        name = n;
        this.age = age;
    }

    void bark() {
      System.out.println("Woof!");
    }
```

```
    void wakeTheNeighbors( ) {
        int i = 50;
        while (i > 0) {
            bark( );
            i = i - 1;
        }
    }

    public static void main(String[ ] args) {
        Dog fido = new Dog("Fido", 5);
        fido.wakeTheNeighbors();
    }


} // ends the class
```

Benefits of OOP

- **Modularity:** The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.
- **Information-hiding:** By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
- **Code re-use:** If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.
- **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.

new keyword

- New operator dynamically allocates memory for an object during run-tme

  class-var= new classname()
- class-var is a variable of the class type being created
- The classname is the name of the class that is being instantiated.
- Class name is followed by parentheses specifies the constructor for the class.
- A constructor defines what occurs when an object of a class is created.
- Most real-world classes explicitly define their own constructors within their class definition.
- If no explicit constructor is specified, then Java will automaticall supply a default constructor

new keyword (Contd)

```
Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object
```

The first line declares **mybox** as a reference to an object of type **Box**. After this line executes, **mybox** contains the value **null**, which indicates that it does not yet point to an actual object. Any attempt to use **mybox** at this point will result in a compile-time error. The next line allocates an actual object and assigns a reference to it to **mybox**. After the second line executes, you can use **mybox** as if it were a **Box** object. But in reality, **mybox** simply holds the memory address of the actual **Box** object. The effect of these two lines of code is depicted in Figure 6-1.

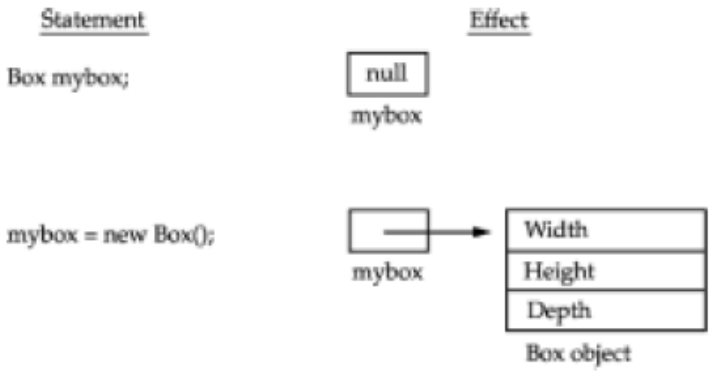


Figure 6.1: Declaring an object of type Box

Methods may contain temporary data

- Data described in a class exists in all objects of that class
  - Example: Every Dog has its own name and age
- A method may contain local *temporary* data that exists only until the method finishes
- Example:
  -
```
void wakeTheNeighbors( ) {
    int i = 50;      // i is a temporary variable
    while (i > 0) {
        bark( );
        i = i - 1;
    }
}
```

Writing and running programs

- When you *write* a program, you are writing classes and all the things that go into classes
- Your program typically contains commands to create objects (that is, "calls" to constructors)
  - Analogy: A class is like a cookie cutter, objects are like cookies.
- When you *run* a program, it creates objects, and those objects interact with one another and do whatever they do to cause something to happen
  - Analogy: *Writing* a program is like writing the rules to a game; *running* a program is like actually playing the game
- You never know how well the rules are going to work until you try them out

Summary

- A program consists of one or more classes
- A class is a description of a kind of object
  - In most cases, it is the objects that do the actual work
- A class describes data, constructors, and methods
  - An object's data is information about that object
  - An object's methods describe how the object behaves
  - A constructor is used to create objects of the class
- Methods (and constructors) may contain temporary data and statements (commands)

THANK YOU