CSE201: Advanced Programming

Lecture 01: Introduction to OOP

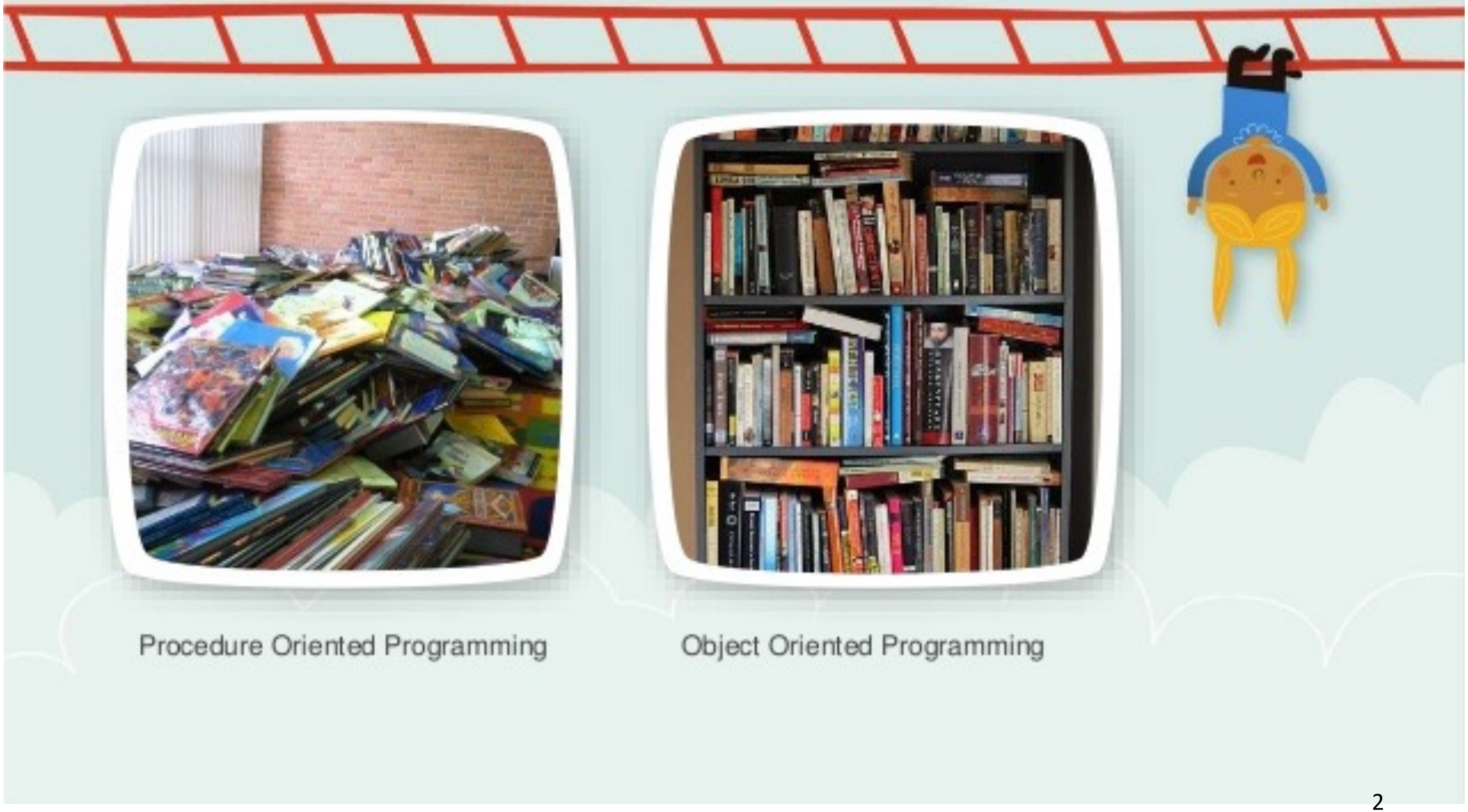
Vivek Kumar

Computer Science and Engineering

IIIT Delhi

vivekk@iiitd.ac.in

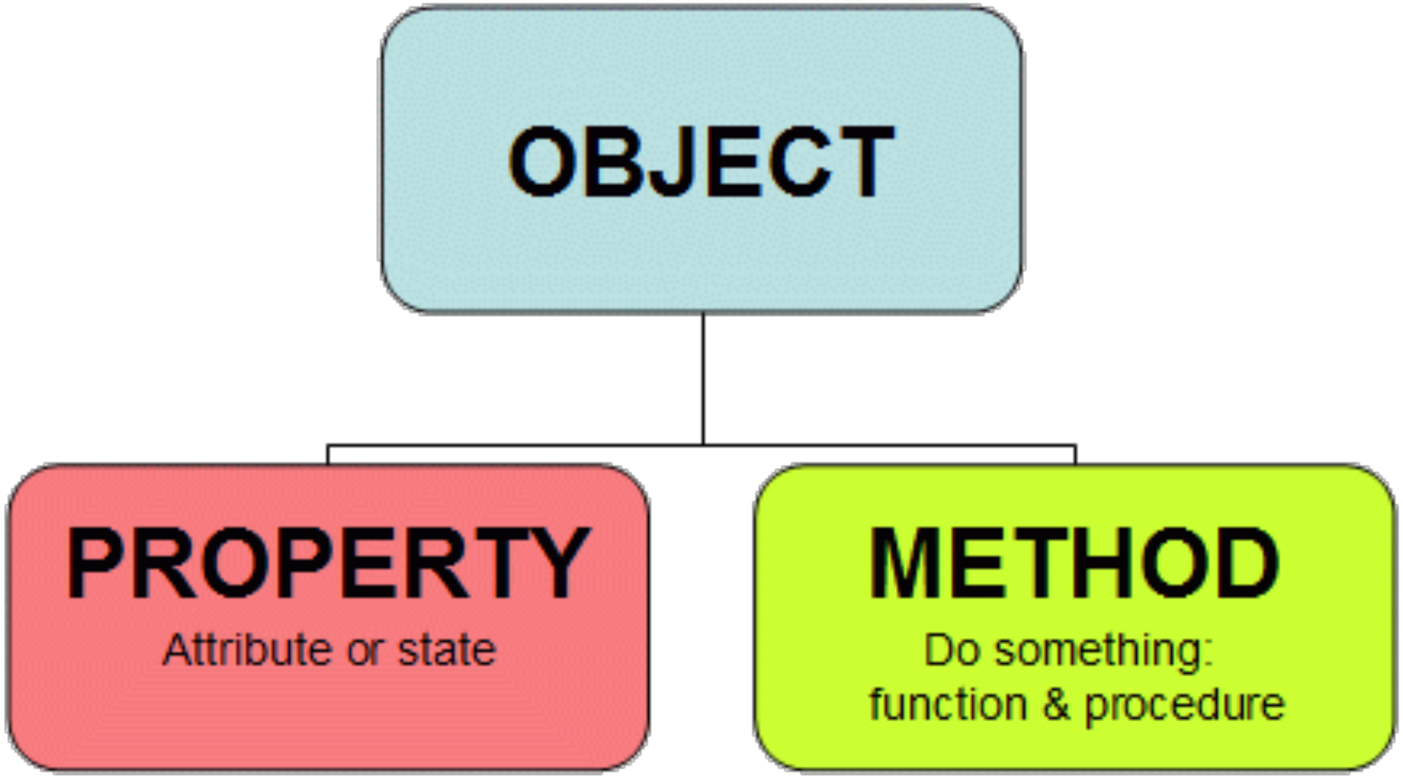
Why Object Oriented Programming?



Procedure Oriented Programming

Object Oriented Programming

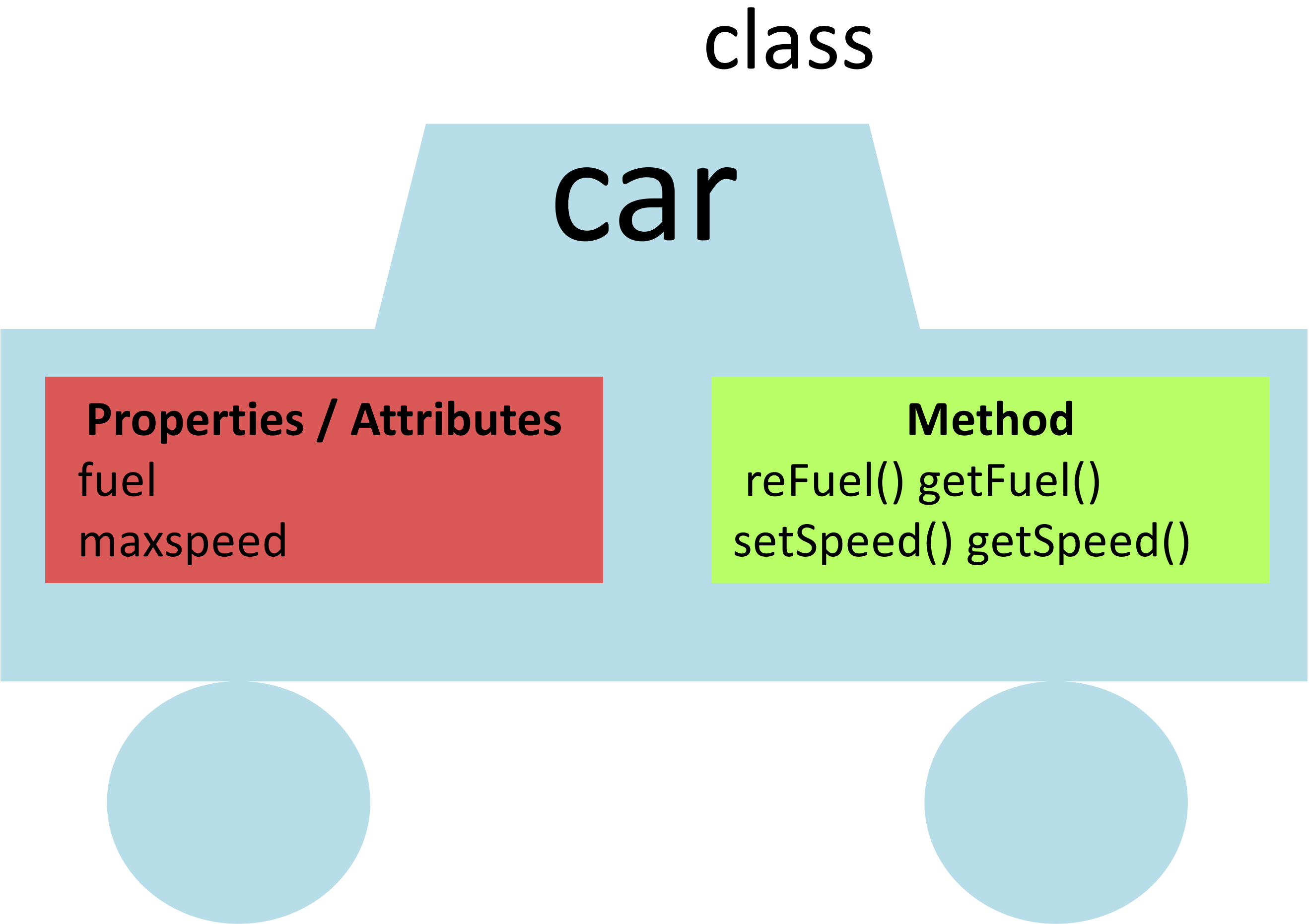
What is OOP?



It is a programming paradigm based on the concept of "objects", which may contain data in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods
(Wikipedia)

© Vivek Kumar

What is OOP?



class

car

Properties / Attributes

fuel

maxspeed

Method

reFuel() getFuel()

setSpeed() getSpeed()

Advantages of OOP

- Code reuse and recycling
  - Objects can easily be reused
- Design benefits
  - Extensive planning phase results better design and lesser flaws
- Software maintenance
  - Easy to incorporate changes in legacy code (e.g., supporting a new hardware)
- Simplicity

© Vivek Kumar

OOP Features

- **Encapsulation**
- Method overloading
- Inheritance
- Abstraction
- Method overriding
- Polymorphism

© Vivek Kumar

Encapsulation

The main thing is How to drive a car ……

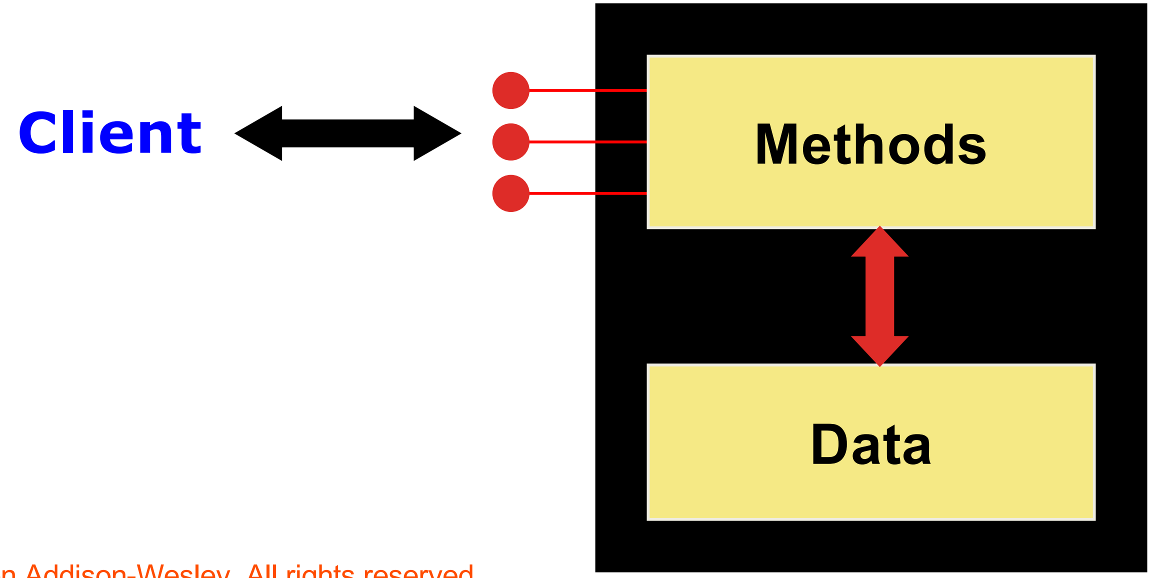
How the car is moving and how the engine is working, this information is hidden.

(**Encapsulation**)

Encapsulation

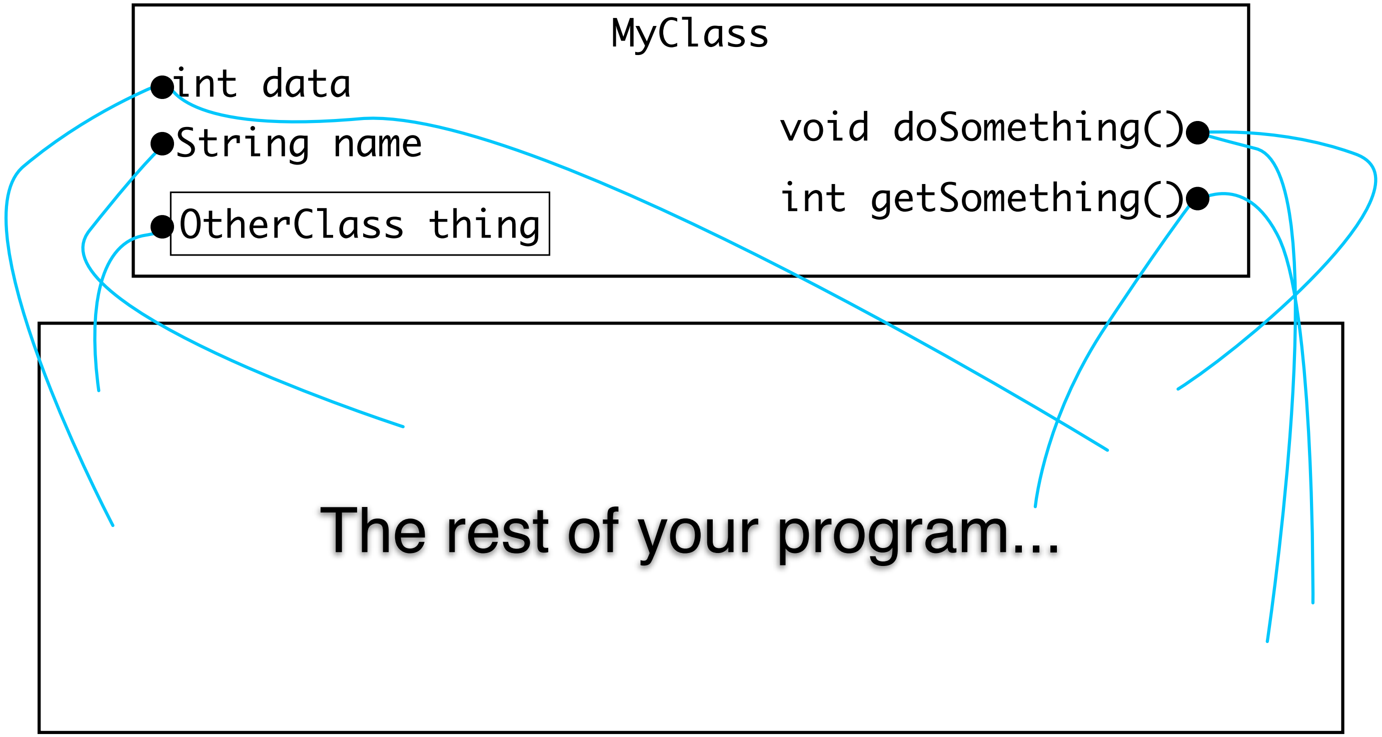
- An encapsulated object can be thought of as a *black box* -- its inner workings are hidden from the client

- The client invokes the interface methods of the object, which manages the instance data



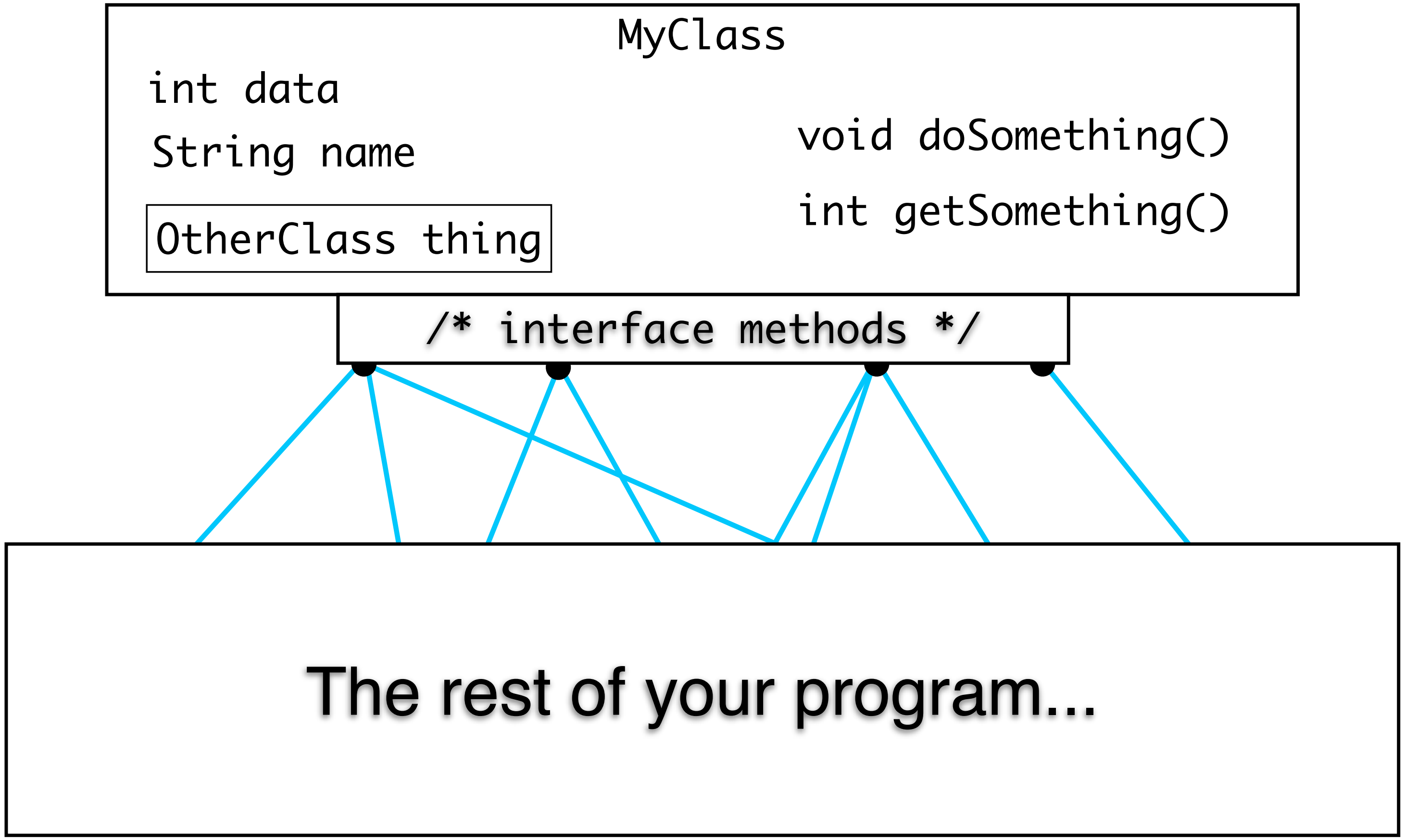
© 2004 Pearson Addison-Wesley. All rights reserved

Class Without Encapsulation



Class Without Encapsulation

Class Supporting Encapsulation



Visibility Modifier

	`public`	`private`
Variables	**Violate encapsulation**	**Enforce encapsulation**
Methods	**Provide services to clients**	**Support other methods in the class**

© 2004 Pearson Addison-Wesley. All rights reserved

Accessors and Mutators

- Because instance data is private, a class usually provides services to access and modify data values
- An *accessor method* returns the current value of a variable
- A *mutator method* changes the value of a variable
- The names of accessor and mutator methods take the form `getX` and `setX`, respectively, where `X` is the name of the value
- They are sometimes called “getters” and “setters”

Wait, but why do we need “setter” when we are talking about restricting accesses to fields from outside world ?

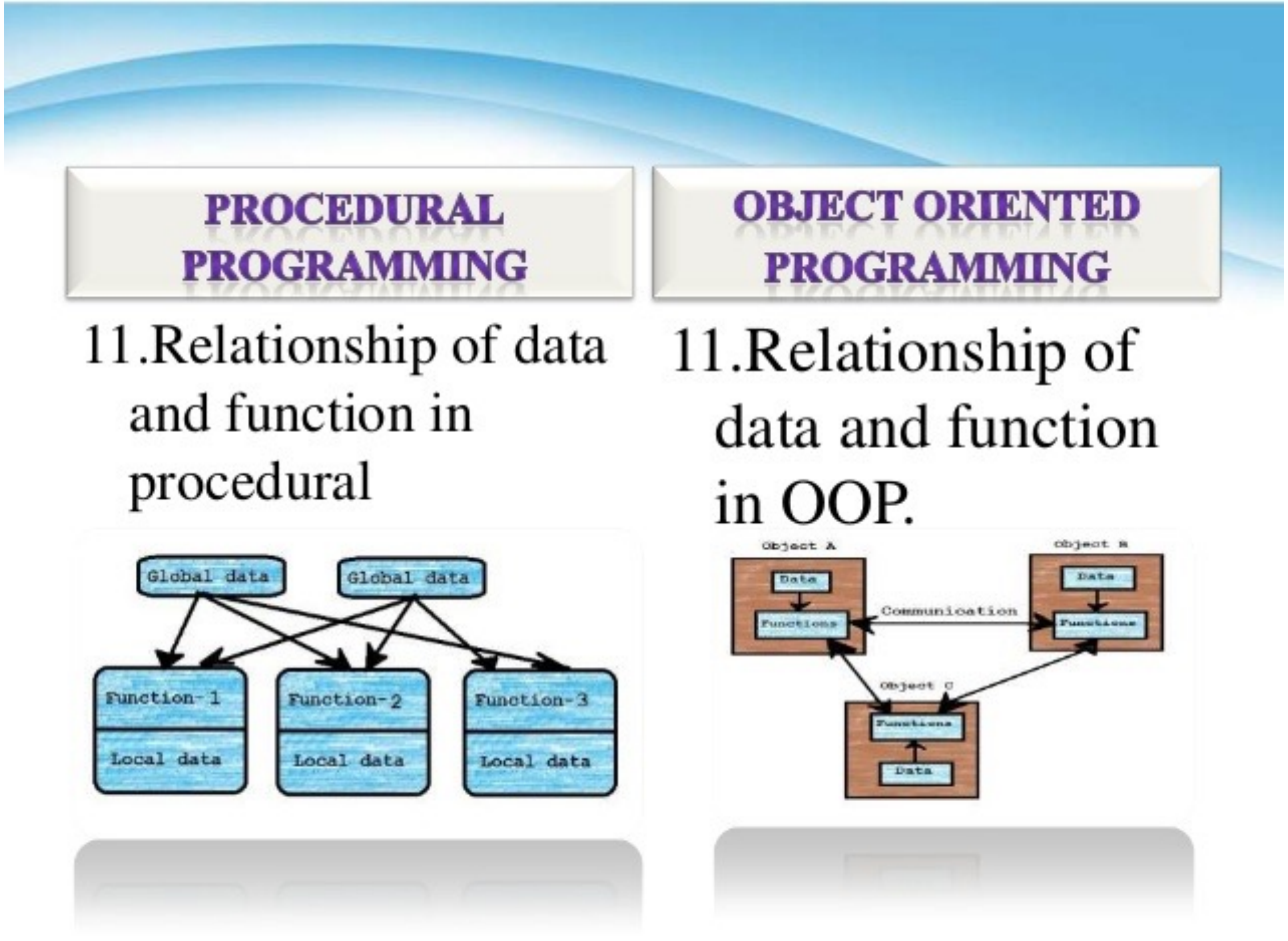
© 2004 Pearson Addison-Wesley. All rights reserved

Mutator Restrictions

- The use of mutators gives the class designer the ability to restrict a client’s options to modify an object’s state
- A mutator is often designed so that the values of variables can be set only within particular limits

© 2004 Pearson Addison-Wesley. All rights reserved

Procedural v/s OOP



PROCEDURAL PROGRAMMING	OBJECT ORIENTED PROGRAMMING
11.Relationship of data and function in procedural	11.Relationship of data and function in OOP.

A Sample Problem

- Write a method that will throw 2 Dice with varying number of sides, a specified amount of times, and reports how many times we got a snake eyes (both dice showing 1)

- For example numSnakeEyes(6, 13, 100) should return the number of snake eyes after throwing a 6 sided Dice and 13 sided Dice 100 times

© 2004 Pearson Addison-Wesley. All rights reserved

Procedural (Structured) Programming Approach

```
static Random rand = new Random();

static int roll(int numFaces) {
    return 1 + rand.nextInt(numFaces);
}

static int numSnakeEyes(int sides1, int sides2, int numThrows) {
    int count = 0;
    for(int i = 0; i < numThrows; i++) {
      int face1 = roll(sides1);
      int face2 = roll(sides2);
      if (face1 == 1 && face2 == 1)
          count++;
    }

    return count;
}
```

© 2004 Pearson Addison-Wesley. All rights reserved

OOP Approach

- In OOP, we first focus on the **main actors**, not how things are done.

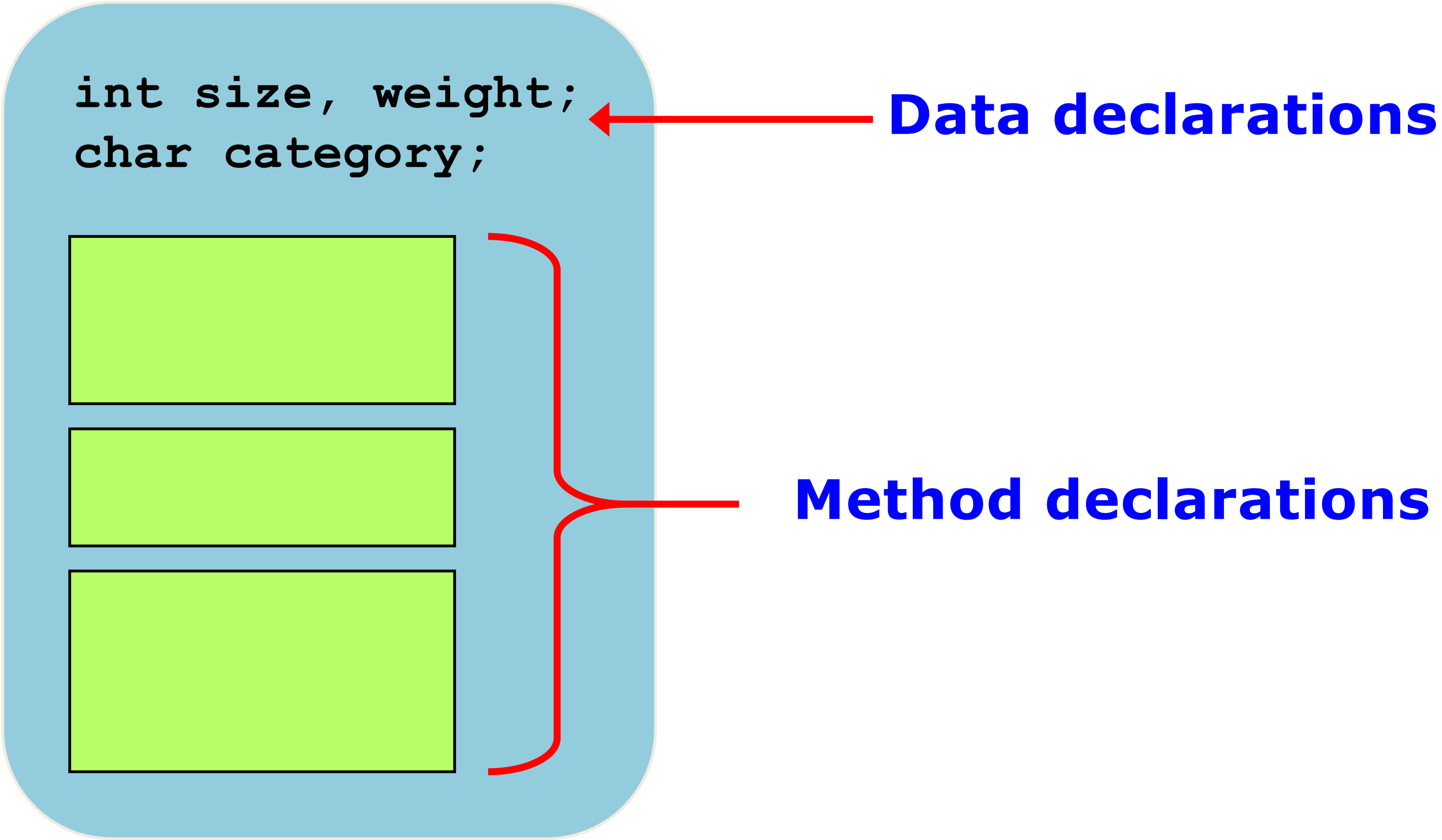
- The main actors here are Dice objects. We need to define a Dice class that captures the *state* and *behavior of a Dice.*

- We can then instantiate as many dice objects as we need for any particular programs

© 2004 Pearson Addison-Wesley. All rights reserved

Classes (Recap)

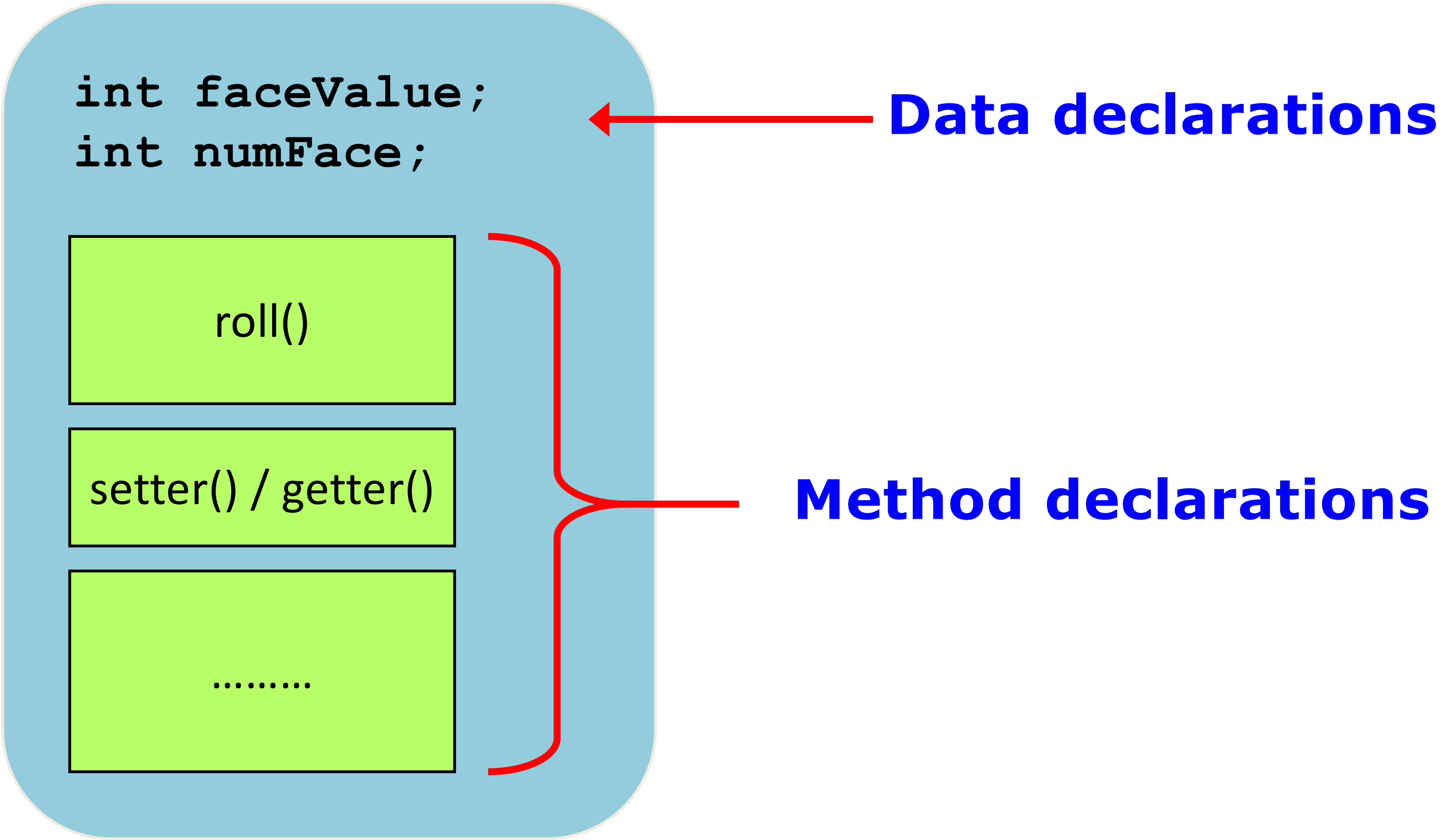
- A class can contain data declarations and method declarations



```
int size, weight;
char category;
```

© 2004 Pearson Addison-Wesley. All rights reserved

Dice Class



```
int faceValue;
int numFace;
```

roll()

setter() / getter()

………

Data declarations

Method declarations

© 2004 Pearson Addison-Wesley. All rights reserved

OOP Approach

```
public class Dice {
    private final int numFaces;  //maximum face value
    private int faceValue;  //current value showing on the dice

    //  Constructor: Sets the initial face value.
    public Dice(int _numFaces)  {
     numFaces = _numFaces;
     roll();
    }

    //  Rolls the dice
    public void roll() {
     faceValue = 1 + rand.nextInt(numFaces);
    }

    //  Face value setter/mutator.
    public void setFaceValue (int value)  {
     if (value <= numFaces)
            faceValue = value;
    }
```

© 2004 Pearson Addison-Wesley. All rights reserved

OOP Approach

```
  //  Face value getter/setter.
  public int getFaceValue() {
        return faceValue;
  }

  //  Face value getter/setter.
  public int getNumFaces() {
        return numFaces;
  }

  //  Returns a string representation of this dice
  public String toString() {
        return “number of Faces “ + numFaces +
        “current face value “ + faceValue);
  }
} // End of Dice class
```

© 2004 Pearson Addison-Wesley. All rights reserved

OOP Approach

The new version

```
static int numSnakeEyes(int sides1, int sides2,
    int numThrows) {
    Die die1 = new Die(sides1);
    Die die2 = new Die(sides2);

    int count = 0;
    for(int i = 0; i < numThrows; i++) {
     die1.roll();
     die2.roll();
     if (die1.getFaceValue == 1 &&
    die2.getFaceValue == 1 )
           count++;
    }

    return count;
}
```

© 2004 Pearson Addison-Wesley. All rights reserved

OOP Approach

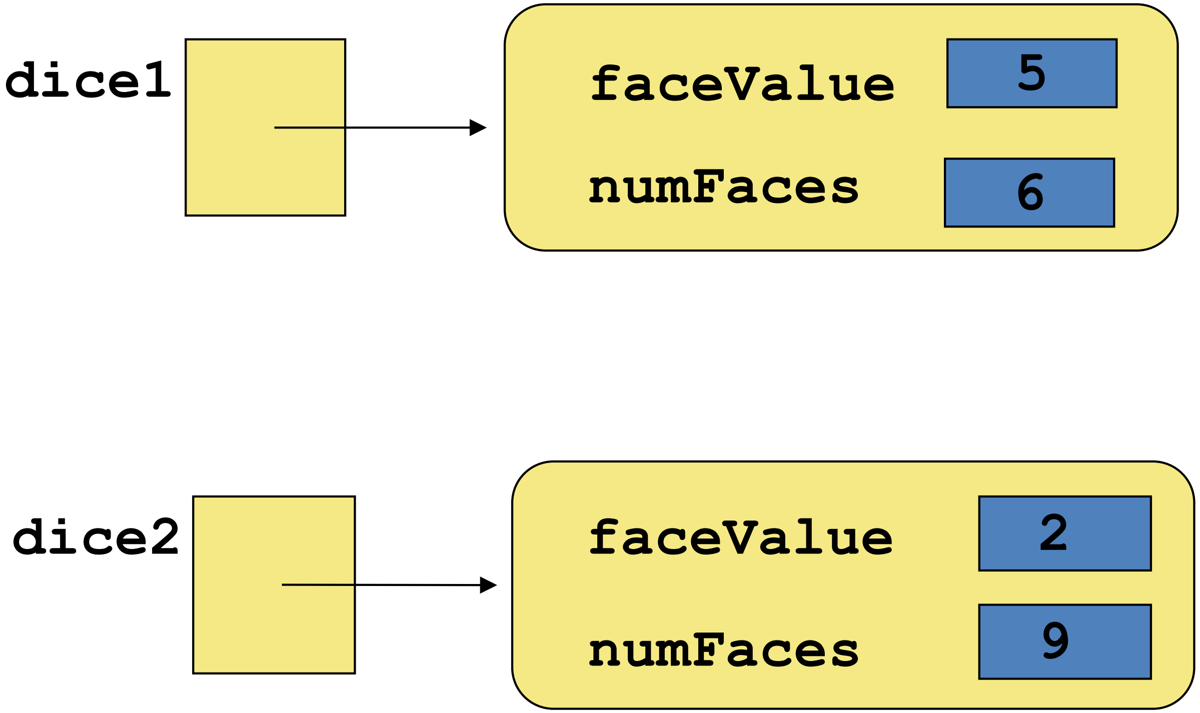
Using Dice class in general

```
Dice dice1, dice2;
int sum;

dice1 = new Dice(7);
dice2 = new Dice(34);

dice1.roll();
dice2.roll();
System.out.println ("Dice One: " + dice1 + ", Dice Two:
    " + dice2);

dice1.roll();
dice2.setFaceValue(4);
System.out.println ("Dice One: " + dice1 + ", Dice Two:
    " + dice2);

sum = dice1.getFaceValue() + dice2.getFaceValue();
System.out.println ("Sum: " + sum);

sum = dice1.roll() + dice2.roll();
System.out.println ("Dice One: " + dice1 + ", Dice Two:
    " + dice2);
System.out.println ("New sum: " + sum);
```

© 2004 Pearson Addison-Wesley. All rights reserved

Instance Data

- We can depict the two `Dice` objects from the `RollingDice` program as follows:



Each object maintains its own `faceValue` and `numFaces` variable, and thus its own state

The toString Method

- All classes that represent objects should define a `toString` method

- The `toString` method returns a character string that represents the object in some way

- It is called automatically when an object is concatenated to a string or when it is passed to the `println` method

© 2004 Pearson Addison-Wesley. All rights reserved

Another Sample Problem

- Coin example
  - Write a program that flips two coins until one of them comes up with heads three times in a row, and report the winner

© 2004 Pearson Addison-Wesley. All rights reserved

Coin Class

```
public class Coin
{
   private final int HEADS = 0;
   private final int TAILS = 1;

   private int face;

  public Coin () {
      flip();
   }
  public void flip () {
      face = (int) (Math.random() * 2);
   }

  public boolean isHeads () {
      return (face == HEADS);
  }
  public String toString() {
      String faceName;
      if (face == HEADS)
         faceName = "Heads";
      else
         faceName = "Tails";
      return faceName;
   }
} // end of class Coin
```

© 2004 Pearson Addison-Wesley. All rights reserved

FlipRace

```
//  Flips two coins until one of them comes up
// heads three times  in a row.
public static void main (String[] args) {
      final int GOAL = 3;
      int count1 = 0, count2 = 0;

      // Create two separate coin objects
      Coin coin1 = new Coin();
      Coin coin2 = new Coin();

      while (count1 < GOAL && count2 < GOAL)
      {
         coin1.flip();
         coin2.flip();

         // Print the flip results (uses Coin's toString method)
         System.out.print ("Coin 1: " + coin1);
         System.out.println ("   Coin 2: " + coin2);

         // Increment or reset the counters
         count1 = (coin1.isHeads()) ? count1+1 : 0;
         count2 = (coin2.isHeads()) ? count2+1 : 0;
      }

      // Determine the winner
      if (count1 < GOAL)
         System.out.println ("Coin 2 Wins!");
      else
         if (count2 < GOAL)
            System.out.println ("Coin 1 Wins!");
         else
            System.out.println ("It's a TIE!");
 } // end of main()
```

© 2004 Pearson Addison-Wesley. All rights reserved

Summary

- What is OOP?
- Encapsulation
  - Visibility modifiers
  - Accessors and mutators
- Simple examples to understand the above concepts

Next Class

- How to identify classes and objects in OOP