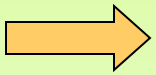
Outline



Writing Classes

Writing Classes

- We've been using predefined classes from the Java API. Now we will learn to write our own classes.
- The class that contains the `main` method is just the starting point of a program
- True object-oriented programming is based on defining classes that represent objects with well-defined characteristics and functionality

Object-Oriented Programming

- An *object* represents an entity in the real world that can be distinctly identified.

- A student, a desk, a circle, a button, a loan can all be viewed as objects.

The State of an Object

- The *state* of an object (also known as its *properties* or *attributes*) is represented by *data fields* with their current values.

- For example:

  – A circle object has a data field **radius**, which is the property that characterizes a circle.

  – A rectangle object has data fields **width** and **height**, which are the properties that characterize a rectangle.

The Behavior of an Object

- The *behavior* of an object (also known as its *actions*) is defined by methods.

- To invoke a method on an object is to ask the object to perform an action.

- You may define a method named **getArea()** for circle objects. A circle object may invoke **getArea()** to return its area.

Class

- Objects of the same type are defined using a common class.
- A class is a template, blueprint, or *contract* that defines what an object's data fields and methods will be.
- An object is an instance of a class.
- You can create many instances of a class.
- Creating an instance is referred to as *instantiation*.

Examples of Classes

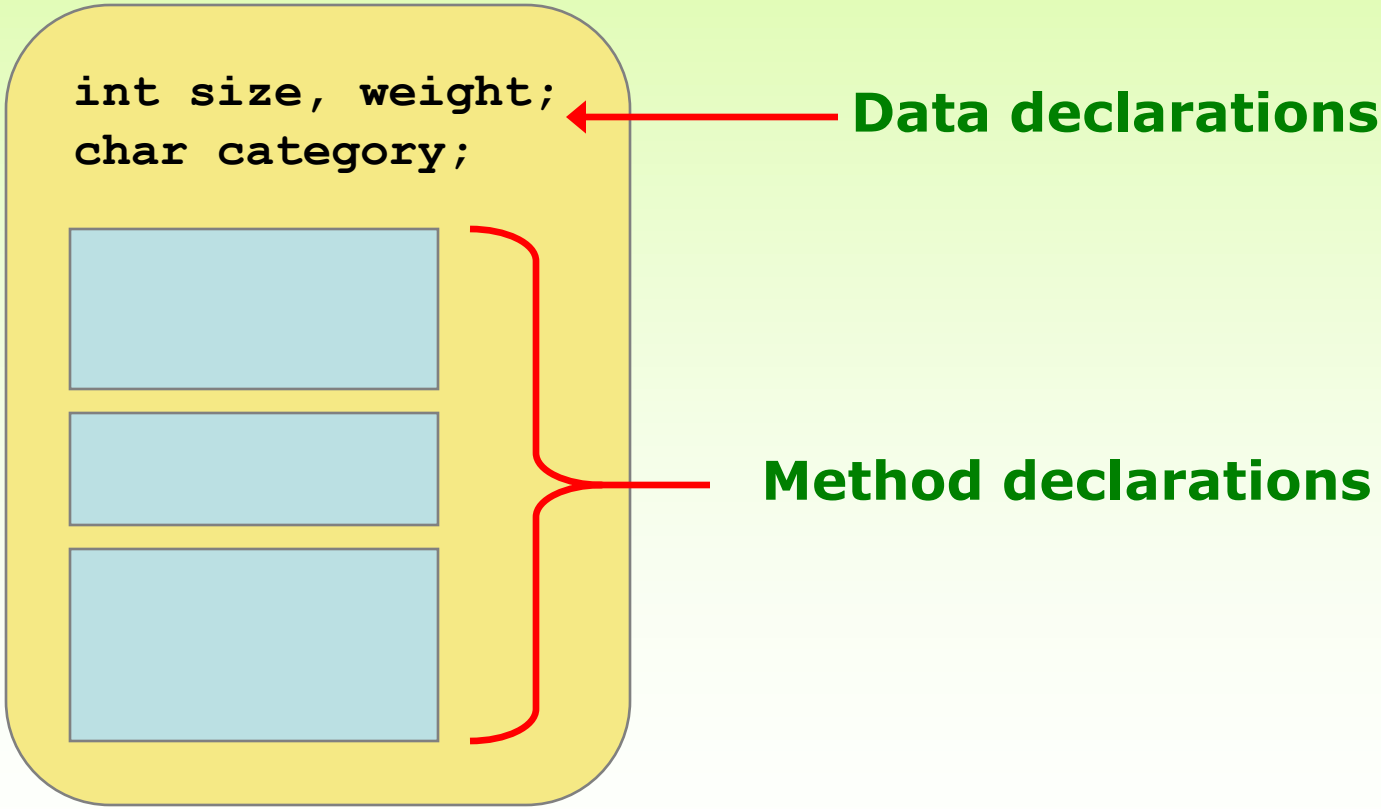
Class	Attributes	Operations
Student	Name Address Major Grade point average	Set address Set major Compute grade point average
Rectangle	Length Width Color	Set length Set width Set color
Aquarium	Material Length Width Height	Set material Set length Set width Set height Compute volume Compute filled weight
Flight	Airline Flight number Origin city Destination city Current status	Set airline Set flight number Determine status
Employee	Name Department Title Salary	Set department Set title Set salary Compute wages Compute bonus Compute taxes

State and Behavior

- Consider a six-sided die
  - It's state can be defined as which face is showing
  - It's primary behavior is that it can be rolled
- We represent a die by designing a class called `Die` that models this state and behavior
  - The class serves as the blueprint for a die object
- We can then instantiate as many die objects as we need for any particular program

Anatomy of a Class

- A class can contain data declarations and method declarations



Example

A class that represents a circle object with radius 1

SimpleCircle.java
TestSimpleCircle.java

Accessing Members of a Class

- **Within a class** you can access a member of the class the same way you would any other variable or method.

- **Outside the class**, a class member is accessed by using the syntax:

  – Referencing variables:

    `objectName.varName` **example:** **`arr.length`**

  – Calling non-static methods on objects:

    `objectName.methodName(params)`

    example: **`str.charAt(0);`**

Constructors

- Constructors are special methods
- A *constructor* is used to set up an object when it is initially created (instantiated)
- A constructor has the same name as the class

Constructors

- A constructor is invoked with the new operator.
  - `Scanner scan = new Scanner(System.in)`
  - `Random randgen = new Random();`

- A constructor should initialize the class variables.
- If the variables are not initialized, default values are
- used.
- A constructor does not have a return type.
- A constructor's identifier (name) is the same as the class it constructs.

Constructors

- Note that a constructor has no return type specified in the method header, not even `void`
- A common error is to put a return type on a constructor
- Each class has a *default constructor* that accepts no parameters

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”

Example

CircleWithConstructors.java

TestCircleWithConstructors.java

Examples

See

MyCircle.java

MyCircleTest.java

Examples

Storing MyCircle objects in an ArrayList:

TestMyCircleArrayList.java

Storing MyCircle objects in an array:

TotalArea.java

The toString Method

- It's good practice to define a `toString` method for a class
- The `toString` method returns a 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

Example

```
//Demonstrates use of toString and the use
  of objects with arraylist
```

Point.java

TestPoint.java

Example: The Die Class

- `Die` class, we might declare an integer called `faceValue` that represents the current value showing on the face
- One of the methods would “roll” the die by setting `faceValue` to a random number between one and six
- The `Die` constructor is used to set the initial face value of each new die object to 1

The Die Class

- We'll want to design the `Die` class so that it is versatile and reusable
- Any given program will probably not use all operations of a given class

Die.java

RollingDice.java

The Die Class

- The `Die` class contains two data values
  - a constant `MAX` that represents the maximum face value
  - an integer `faceValue` that represents the current face value
- The `roll` method uses the `random` method of the `Math` class to determine a new face value
- There are also methods to explicitly set and retrieve the current face value at any time

```
//********************************************************************
//  RollingDice.java       Author: Lewis/Loftus
//
//  Demonstrates the creation and use of a user-defined class.
//********************************************************************

public class RollingDice
{
   //-----------------------------------------------------------------
   //  Creates two Die objects and rolls them several times.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Die die1, die2;
      int sum;

      die1 = new Die();
      die2 = new Die();

      die1.roll();
      die2.roll();
      System.out.println ("Die One: " + die1 + ", Die Two: " + die2);
```

continue

continue

```
        die1.roll();
        die2.setFaceValue(4);
        System.out.println ("Die One: " + die1 + ", Die Two: " + die2);

        sum = die1.getFaceValue() + die2.getFaceValue();
        System.out.println ("Sum: " + sum);

        sum = die1.roll() + die2.roll();
        System.out.println ("Die One: " + die1 + ", Die Two: " + die2);
        System.out.println ("New sum: " + sum);
    }
}
```

continue

```
        die1.roll();
        die2.setFaceV
        System.out.pr                                    , Die Two: " + die2);

        sum = die1.ge                                    lue();
        System.out.pr

        sum = die1.roll() + die2.roll();
        System.out.println ("Die One: " + die1 + ", Die Two: " + die2);
        System.out.println ("New sum: " + sum);
    }
}
```

Sample Run

```
Die One: 5, Die Two: 2
Die One: 1, Die Two: 4
Sum: 5
Die One: 4, Die Two: 2
New sum: 6
```

```
//********************************************************************
//  Die.java       Author: Lewis/Loftus
//
//  Represents one die (singular of dice) with faces showing values
//  between 1 and 6.
//********************************************************************

public class Die
{
   private final int MAX = 6;  // maximum face value

   private int faceValue;  // current value showing on the die

   //-----------------------------------------------------------------
   //  Constructor: Sets the initial face value.
   //-----------------------------------------------------------------
   public Die()
   {
      faceValue = 1;
   }
```

continue

continue

```
//-----------------------------------------------------------------
//  Rolls the die and returns the result.
//-----------------------------------------------------------------
public int roll()
{
   faceValue = (int)(Math.random() * MAX) + 1;
   return faceValue;
}

//-----------------------------------------------------------------
//  Face value mutator.
//-----------------------------------------------------------------
public void setFaceValue (int value)
{
   faceValue = value;
}

//-----------------------------------------------------------------
//  Face value accessor.
//-----------------------------------------------------------------
public int getFaceValue()
{
   return faceValue;
}
```

continue

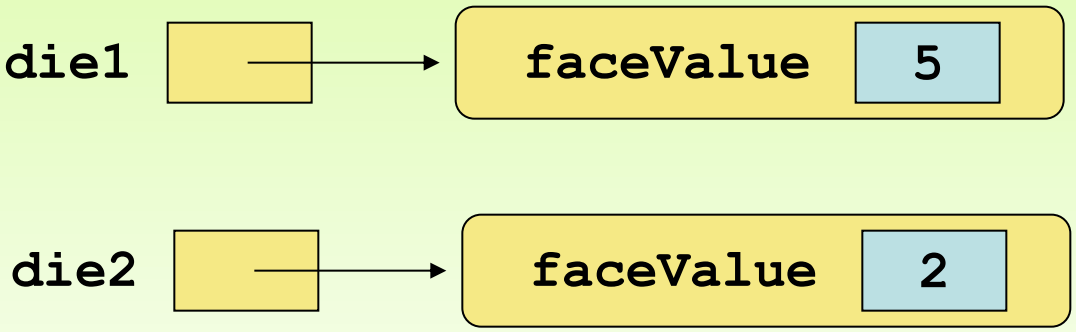
continue

```
    //-----------------------------------------------------------------
    //  Returns a string representation of this die.
    //-----------------------------------------------------------------
    public String toString()
    {
        String result = Integer.toString(faceValue);

        return result;
    }
}
```

Instance Data

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



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

Instance Data

- A variable declared at the class level (such as `faceValue`) is called *instance data*
- Each instance (object) has its own instance variable
- A class declares the type of the data, but it does not reserve memory space for it
- Each time a `Die` object is created, a new `faceValue` variable is created as well
- The objects of a class share the method definitions, but each object has its own data space
- That's the only way two objects can have different states