Chapter 9 continued

- This chapter is PACKED! Please make sure you read it!
Static Modifier

- In our Circle example, the variables radius, name, xPos, yPos are all variables for a specific *instance*. For example, *circle1’s* radius, name, etc

- If you want to share data between *all* the circle objects you make, you can use a *static variable*

- These are stored in a common location in memory where all instances can access them.

- Changes made from any instance of a class are reflected in every instance.

- In java you can use *static variables* as well as *static methods* and *static methods* can be called without a specific instance of a class
Static Modifier - example
```java
public class TestCircle_InClass {
    public static void main(String[] args) {
        System.out.println(CircleWithStatic.getNumberOfObjects());
        CircleWithStatic circle1 = new CircleWithStatic();
        CircleWithStatic circle2 = new CircleWithStatic();
        CircleWithStatic circle3 = new CircleWithStatic();
        System.out.println(circle1.numberOfObjects);
        System.out.println(CircleWithStatic.numberOfObjects);
        System.out.println(CircleWithStatic.getNumberOfObjects());
    }
}

public class CircleWithStatic {
    /** The radius of this circle */
    static int numberOfObjects = 0;
    double radius;
    String name;
    int xPos = 0;
    int yPos = 0;

    /** Construct a circle object */
    CircleWithStatic() {
        numberOfObjects++;
    }

    /** Construct a circle object */
    CircleWithStatic(double newRadius) {
        radius = newRadius;
        numberOfObjects++;
    }

    CircleWithStatic(double newRadius, int xIn, int yIn) {
        radius = newRadius;
        xPos = xIn;
        yPos = yIn;
        numberOfObjects++;
    }

    /** Return the area of this circle */
    double getArea() {
        return radius * radius * Math.PI;
    }

    /** Return the perimeter of this circle */
    double getPerimeter() {
        return 2 * radius * Math.PI;
    }

    /** Set new radius for this circle */
    void setRadius(double newRadius) {
        radius = newRadius;
    }

    void printYpos() {
        System.out.println("The Y position is: " + yPos);
    }

    int getXpos() {
        return xPos;
    }

    static int getNumberOfObjects() {
        return numberOfObjects;
    }
}
```
Static Method Rules

• It’s best to call static methods from the classname as opposed to an instance of a class. So instead of:

  • myCircle1.getNumberOfObjects(), use Circle.getNumberOfObjects()
Static Method Rules

• An instance method can invoke or access both instance and static data fields and methods

• A static method can only invoke or access static data fields and methods

• This is because the static (class) methods don’t know about the instance objects
Static Method Rules
Visibility Modifiers

• The *public* modifier allows classes, methods and data fields to be accessed *from other classes*

• The *private* modifier allows methods and data fields to *only be accessed from within the same class*

• The default (if no visibility modifier is used) is *package-private* or *package-access* meaning any other class *in the same package can access them*
Private Modifier

```java
package p1;

public class C1 {
    public int x;
    int y;
    private int z;

    public void m1() {
    }
    void m2() {
    }
    private void m3() {
    }
}
```

```java
package p1;

public class C2 {
    void aMethod() {
        C1 o = new C1();
    }
}
```

```java
package p2;

public class C3 {
    void aMethod() {
        C1 o = new C1();
    }
}
```
Private Modifier

```java
package p1;

public class C1 {
    public int x;
    int y;
    private int z;

    public void m1() {
    }
    void m2() {
    }
    private void m3() {
    }
}

package p1;

public class C2 {
    void aMethod() {
        C1 o = new C1();
        can access o.x;
        can access o.y;
        cannot access o.z;

        can invoke o.m1();
        can invoke o.m2();
        cannot invoke o.m3();
    }
}

package p2;

public class C3 {
    void aMethod() {
        C1 o = new C1();
        can access o.x;
        cannot access o.y;
        cannot access o.z;

        can invoke o.m1();
        cannot invoke o.m2();
        cannot invoke o.m3();
    }
}
```
Private Modifier Example
public class CircleWithPrivate {
    /** The radius of this circle */
    static int numberOfObjects = 0;
    double radius;
    private String name;
    int xPos = 0;
    int yPos = 0;

    /** Construct a circle object */
    CircleWithPrivate() {
        numberOfObjects++;
    }

    /** Construct a circle object */
    CircleWithPrivate(double newRadius) {
        radius = newRadius;
        numberOfObjects++;
    }

    CircleWithPrivate(double newRadius, int xIn, int yIn) {
        radius = newRadius;
        xPos = xIn;
        yPos = yIn;
        numberOfObjects++;
    }

    /** Return the area of this circle */
    double getArea() {
        return radius * radius * Math.PI;
    }

    /** Return the perimeter of this circle */
    double getPerimeter() {
        return 2 * radius * Math.PI;
    }

    /** Set new radius for this circle */
    void setRadius(double newRadius) {
        radius = newRadius;
    }

    void printYpos() {
        System.out.println("The Y position is: " + yPos);
    }

    int getXpos() {
        return xPos;
    }

    static int getNumberOfObjects() {
        return numberOfObjects;
    }

    String getName() {
        return name;
    }
}

public class TestCircle_InClass {
    public static void main(String[] args) {
        System.out.println(CircleWithStatic.getNumberOfObjects());
        CircleWithStatic circle1 = new CircleWithStatic();
        CircleWithStatic circle2 = new CircleWithStatic();
        CircleWithStatic circle3 = new CircleWithStatic();
        System.out.println(circle1.numberOfObjects);
        System.out.println(CircleWithStatic.numberOfObjects);
        System.out.println(CircleWithStatic.getNumberOfObjects());
        CircleWithPrivate circle4 = new CircleWithPrivate();
        System.out.println(circle4.getName());
    }
}
Private Modifier

- Note: most of the time, you’ll want public constructors

- An exception is if you only have static methods, and don’t want a user to create an instance of an object (like Math class, for example)
Data Field Encapsulation

- Using private data fields like this is called *data field encapsulation*
- It’s extremely useful to protect data from being messed with
- Keep our code cleaner and easier to debug
Data Field Encapsulation

• You can restrict access to variables by setting variables private and using getter and setter methods for those variables

• Let’s make sure all our variables have getters and setters
Passing objects to methods

• Works the same as passing an array - we actually pass the *reference* to the array or object.
Practice - Passing objects to methods

• Let’s write a method in our test class that takes a Circle object as a parameter and prints out the radius of that circle
Array of Objects

- You can use the objects we create in an array as well!
  - Circle[] circleArray = new Circle[7]

- To initialize with new objects, you could loop through the array and create a new object for each element:

```java
for (int i = 0; i < circleArray.length; i++) {
    circleArray[i] = new Circle();
}
```
public class TestCircle_InClass {

    public static void main(String[] args) {
        System.out.println(CircleWithStatic.getNumberOfObjects());

        CircleWithStatic circle1 = new CircleWithStatic();
        CircleWithStatic circle2 = new CircleWithStatic();
        CircleWithStatic circle3 = new CircleWithStatic();

        System.out.println(circle1.numberOfObjects);
        System.out.println(CircleWithStatic.numberOfObjects);
        System.out.println(CircleWithStatic.getNumberOfObjects());

        CircleWithPrivate circle4 = new CircleWithPrivate();
        System.out.println(circle4.getName());

        CircleWithStatic[] circleArray = new CircleWithStatic[10];

        for (int i = 0; i < circleArray.length; i++) {
            circleArray[i] = new CircleWithStatic();
        }

        for (int i = 0; i < circleArray.length; i++) {
            System.out.println(circleArray[i].getArea());
        }
    }
}
Immutable Objects

• Once the object is created, can’t be changed

• The *String* class is an example of an immutable object

• **All data fields must be private**

• No setters allowed

• No getters that return a reference to a data field that could be changed
Variable Scope

- Local variables are declared in a method, class variables are declared outside of all the methods but inside the class.

- Scope of local variables is just in the method in which it’s declared, scope of class variables are the whole class.
public class F {
    private int x = 0; // Instance variable
    private int y = 0;

    public F() { }

    public void p() {
        int x = 1; // Local variable
        System.out.println("x = " + x);
        System.out.println("y = " + y);
    }
}
public class testF {

    public static void main(String[] args) {

        F f = new F();
        f.p();
    }
}

Hidden Variables

- If a local variable is declared with the same name as a class variable, the local will be used.
- The class variable is “hidden”
Hidden Variables

- To access hidden variables, use the *this* keyword
- *this* is a reference to current instance of the object
public class F {
    private int x = 0; // Instance variable
    private int y = 0;

    public F() {
    }

    public void p() {
        int x = 1; // Local variable
        System.out.println("x = " + this.x);
        System.out.println("y = " + y);
    }
}
**this** with constructors

- You can use the *this* keyword to call other constructors within the class
- Easier to maintain and understand the code
- You use **this()** instead of **ClassName()** to call the constructor
public class CircleThis {
    /** The radius of this circle */
    double radius;
    int xPos = 0;
    int yPos = 0;

    /** Construct a circle object */
    CircleThis() {
        this(1.0);
    }

    /** Construct a circle object */
    CircleThis(double newRadius) {
        this(newRadius, 10, 10);
    }

    CircleThis(double newRadius, int xIn, int yIn) {
        radius = newRadius;
        xPos = xIn;
        yPos = yIn;
    }
}
Practice - Book class

• Write a class that can be used to describe books including the following information:
  • Title
  • Author
  • Year published
  • Page count
  • Genre
  • A running count of how many books have been entered

• Then write a test class that asks a user to enter the number of books to store, and let’s the user enter each piece of information for each book. Write a method that prints out a summary for each book entered.

• When writing your class, be sure to use encapsulation (getters and setters) and private and static modifiers when appropriate