

Java Tutorial: Working with Objects and Classes

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 688R, Spring Semester 2023

March 5, 2023

Overview

- 1 Working with Objects
- 2 Encapsulation and Data Hiding
- 3 Relationships Among Classes
- 4 Association Relationships
- 5 Inheritance Mechanisms
- 6 Composition of Object Models
- 7 Applications

Part 1

Working with Objects

Object-based Software

Basic Assumptions:

- **Everything** is an **object**.
- New kinds of objects can be created by making a package containing other existing objects.
- Objects have relationships for other types of objects.
- Objects have type.
- Object communicate via message passing – all objects of the same type can receive and send the same kinds of messages.
- Objects can have **executable behavior**.
- Objects can be design to **respond to** occurrences and **events**.
- Systems will be created through a **composition (assembly) of objects**.

Working with Objects and Classes

Working with Objects and Classes:

- Collections of objects share similar traits (e.g., data, structure, behavior).
- Collections of objects will form relationships with other collections of objects.

Definition of a Class

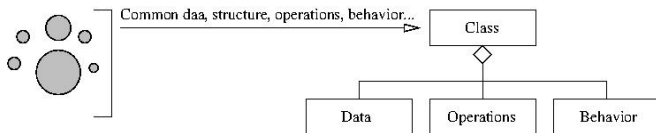
A **class** is a **specification** (or blueprint) of an object's structure and behavior.

Definition of an Object

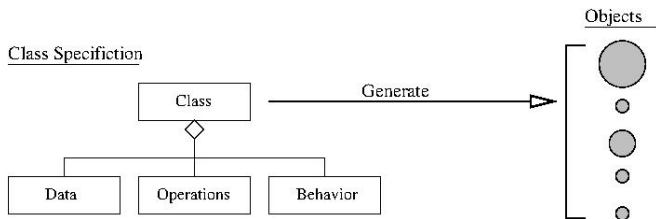
An **object** is an **instance** of a class.

Working with Objects and Classes

From Collections of Objects to Classes:



Generation of Objects from Class Specifications:



Working with Objects and Classes

Key Design Tasks

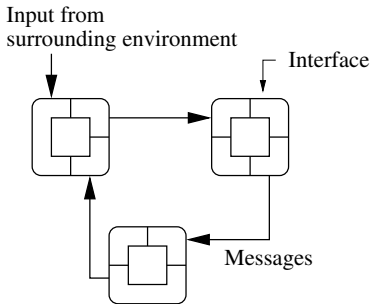
- Identify **objects** and their **attributes** and **functions**,
- Establish **relationships** among the objects,
- Establish the **interfaces** for each object,
- Implement and test the individual objects,
- Assemble and test the system.

Implicit Assumptions:

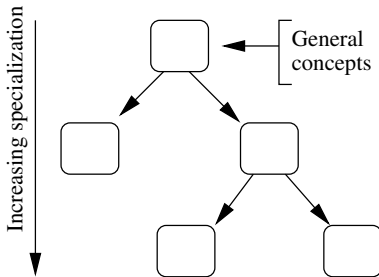
- **Manual synthesis** of the **object model** is realistic for systems that have a **modest number of elements and relationships**.
- As the dimensionality of the problem increases some form of **automation** will be needed to **discover elements** and **relationships**.

Working with Objects and Classes

Organizational and Efficiency Mechanisms:



Network of Communicating Objects



Problem Domain Concepts organized into a Class Hierarchy.

Example 1. Working with Points

A Very Simple Class in Java

```
1     public class Point {
2         int x, y;
3
4         public Point ( int x, int y ) {
5             this.x = x; this.y = y;
6         }
7     }
```

Creating an Object

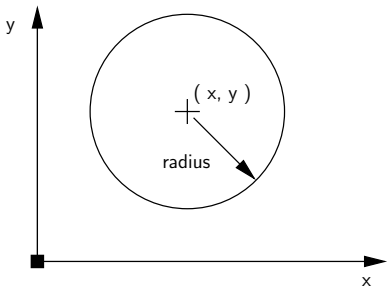
```
8         Point first = new Point ( 1, 2 );
9         Point second = new Point ( 2, 5 );
```

Accessing and Printing the attributes on an Object

```
10        System.out.printf(" first point (x,y) = (%2d, %2d)\n", first.x, first.y );
11        System.out.printf("second point (x,y) = (%2d, %2d)\n", second.x, second.y );
```

Example 2. Working with Circles

A circle can be described by the (x, y) position of its center and by its radius.



There are numerous things we can do with circles:

- Compute their circumference, perimeter or area,
- Check if a point is inside a circle.

Example 2. Working with Circles

```
1  /*
2  *  =====
3  *  Circle(): Basic implementation of a circle program.
4  *
5  *  Written by: Mark Austin                      February, 2019
6  *  =====
7  */
8
9  import java.lang.Math.*;
10
11 public class Circle {
12     public double dX, dY, dRadius;
13
14     // Constructor
15
16     public Circle () {}
17
18     public Circle( double dX, double dY, double dRadius ) {
19         this.dX = dX;
20         this.dY = dY;
21         this.dRadius = dRadius;
22     }
23
24     // Compute the circle area ....
25
26     public double Area() {
27         return Math.PI*dRadius*dRadius;
28     }
```

Example 2. Working with Circles

```

29
30 // Copy circle parameters to a string format ...
31
32 public String toString() {
33     return "(x,y) = (" + dX + "," + dY + "): Radius = " + dRadius;
34 }
35
36 // -----
37 // Exercise methods in class Circle ...
38 // -----
39
40 public static void main( String [] args ) {
41
42     System.out.println("Exercise methods in class Circle");
43     System.out.println("=====");
44
45     Circle cA = new Circle();
46     cA.dX = 1.0; cA.dY = 2.0; cA.dRadius = 3.0;
47
48     Circle cB = new Circle( 1.0, 2.0, 2.0 );
49
50     System.out.printf("Circle cA : %s\n", cA.toString() );
51     System.out.printf("Circle cA : Area = %5.2f\n", cA.Area() );
52     System.out.printf("Circle cB : %s\n", cB );
53     System.out.printf("Circle cB : Area = %5.2f\n", cB.Area() );
54 }
55 }

```

Example 2. Working with Circles

Script of Program Input and Output

Exercise methods in class Circle

=====

Circle cA : (x,y) = (1.0,2.0): Radius = 3.0

Circle cA : Area = 28.27

Circle cB : (x,y) = (1.0,2.0): Radius = 2.0

Circle cB : Area = 12.57

Points to note:

- Objects are created with [constructor methods](#). The line:

```
public Circle () {}
```

is the default constructor. It creates circle objects with all of the circle attribute values initialized to zero.

Example 2. Working with Circles

More points to note:

- The next three statements use the dot notation (.) to manually initialize the (x,y) coordinates of the circle center and its radius.
- A second constructor method:

```
public Circle( double dX, double dY, double dRadius ) {  
    }  
}
```

creates a circle object and initializes the circle attribute values in one line.

- Statements of the form `this.dX = dX` take the value of `dX` passed to the constructor method and assign it to the attribute `dX` associated with [this object](#).

Accessing Object Data and Object Methods

Now that we have created an object, we can use its data fields. The **dot operator (.)** is used to access the different public variables of an object.

Example 1

```
Circle cA = new Circle();  
cA.dX = 1.0;  
cA.dY = 2.0;  
cA.dRadius = 3.0;
```

To **access the methods of an object**, we use the same syntax as accessing the data of the object, i.e., **the dot operator (.)**.

Accessing Object Methods

Example 2

```
Circle cA = new Circle();  
cA.dRadius = 2.5;  
double dArea = cA.getArea();
```

Notice that we did not write `dArea = getArea(cA);`

Example 3

Let `a`, `b`, `c`, and `d` be complex numbers. To compute $a*b + c*d$ we write

```
a = new Complex(1,1); .. etc ..  
  
Complex sum = a.Mult(b).Add( c.Mult(d) );
```


Encapsulation and Data Hiding

Encapsulation and Data Hiding

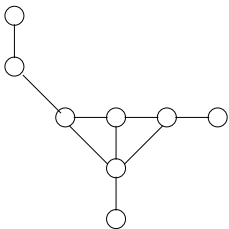
Definition of Aggregation

- Aggregation is the grouping of components into a package.
- Aggregation does not imply that the components are hidden or inaccessible. It merely implies that the components are part of a whole.

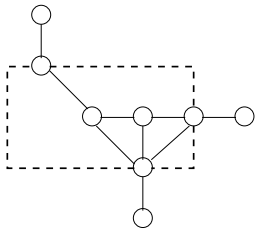
Definition of Encapsulation

- Encapsulation is a much stronger form of organization.
- Encapsulation forces users of a system to deal with it as an abstraction (e.g., a black box) with well-defined interfaces that define what the entity is, what it does, and how it should be used.
- The only way to access an object's state is to send it a message that causes one of the object's internal methods to execute.

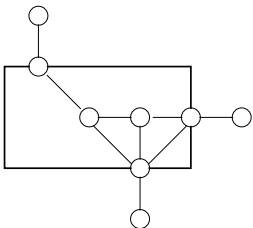
Encapsulation and Data Hiding



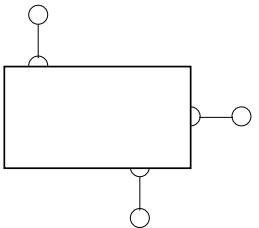
Unstructured Components



Aggregation



Designer's view of Aggregation



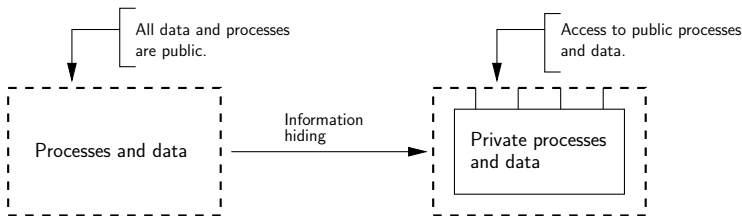
Encapsulation – User's view of Abstraction

Encapsulation and Data Hiding

Principle of Information Hiding

The principle of information hiding states that **information which is likely to change** (e.g., over the lifetime of a software/systems package) should be **hidden inside a module**.

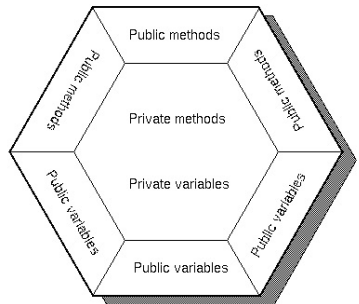
Application. Process for Implementation of Information Hiding.



Encapsulation and Data Hiding

Graphical Representation of a Class

Graphical representation of a Class



The object wrapping **protects the object code** from unintended access by other code.

Encapsulation and Data Hiding

In object-oriented terminology, and particularly in Java,

- The wrapper object is usually called a **class**, the functions inside the class are called **private methods**,
- The data inside the class are **private variables**.
- **Public methods** are the interface functions for the outside world to access your private methods.

Implementation. The keyword **private** in:

```
public class Point {  
    private int x, y;  
    ....  
}
```

restricts to scope of `x` and `y` to lie inside the boundary of `Point` objects.

Encapsulation and Data Hiding

Access to a point's coordinates is controlled through the public methods:

```
public int  getX() {  
    return x;  
}  
public void setX(int x) {  
    this.x = x;  
}
```

Example 2. Revised Circle Program

Revised circle program where **data and circle properties** can only be accessed through an **interface**.

```

1  /*
2  *  =====
3  *  Circle(): Implementation of the Circle class where data and circle
4  *  properties can only be accessed through an interface.
5  *
6  *  Written by: Mark Austin                                February, 2019
7  *  =====
8  */
9
10 import java.lang.Math.*;
11
12 public class Circle {
13     protected double dX, dY, dRadius;
14
15     // Constructor
16
17     public Circle () {}
18
19     public Circle( double dX, double dY, double dRadius ) {
20         this.dX = dX;
21         this.dY = dY;
22         this.dRadius = dRadius;
23     }
24
25     // Compute the circle area ....

```


Example 2. Revised Circle Program

```

26
27     private double Area() {
28         return Math.PI*dRadius*dRadius;
29     }
30
31     // Create public interface for variables and area computation....
32
33     public void setX (double dX) {
34         this.dX = dX;
35     }
36
37     public double getX () {
38         return dX;
39     }
40
41     ... details for setY() and getY() removed ...
42
43     public void setRadius (double dRadius ) {
44         this.dRadius = dRadius;
45     }
46
47     public double getRadius () {
48         return dRadius;
49     }
50
51     public double getArea() {
52         return Area();
53     }
54
55     // Copy circle parameters to a string format ...

```

Example 2. Revised Circle Program

```

56
57     public String toString() {
58         return "(x,y) = (" + dX + "," + dY + "): Radius = " + dRadius;
59     }
60
61     // -----
62     // Exercise methods in class Circle ...
63     // -----
64
65     public static void main( String [] args ) {
66
67         System.out.println("Exercise methods in class Circle");
68         System.out.println("=====");
69
70         Circle cA = new Circle();
71         cA.setX(1.0);
72         cA.setY(2.0);
73         cA.setRadius(3.0);
74
75         Circle cB = new Circle( 1.0, 2.0, 2.0 );
76
77         System.out.printf("Circle cA : %s\n", cA.toString() );
78         System.out.printf("Circle cA : Area = %5.2f\n", cA.getArea() );
79
80         System.out.printf("Circle cB : %s\n", cB );
81         System.out.printf("Circle cB : Area = %5.2f\n", cB.getArea() );
82     }
83 }

```

Example 2. Revised Circle Program

Points to note:

- Use of the keyword `protected` in:

```
protected double dX, dY, dRadius;
```

restricts access of `dX`, `dY` and `dRadius` to **methods within Circle** and **any subclass of Circle**.

- The methods `getX()` and `setX()`, etc, create a **public interface** for Circle.
- By convention, the **`toString()` method** creates and returns a string description of the objects contents. And it can be **called in two ways** as demonstrated at the bottom of `main()`. The fragment of code **`cA.toString()`** will return a string which will be matched against the **`%s`** format specification. However, **`cB`** also calls **`toString()`** and is shorthand for **`cB.toString()`**.