Thinking in Objects

Problem Solving using OOP

The *this* Keyword
Reference the Hidden Data Fields

The <u>this</u> keyword is the name of a reference that refers to an object itself. One common use of the <u>this</u> keyword is reference a class's *hidden data fields*.

```java
public class Foo {
  private int i = 5;
  private static double k = 0;

  void setI(int i) {
    this.i = i;
  }

  static void setK(double k) {
    Foo.k = k;
  }
}
```

```
Suppose that f1 and f2 are two objects of Foo.

Invoking f1.setI(10) is to execute
    this.i = 10, where this refers f1

Invoking f2.setI(45) is to execute
    this.i = 45, where this refers f2
```
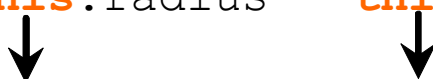
Calling Overloaded Constructor

Another common use of the <u>this</u> keyword to enable a constructor to invoke another constructor of the same class.

```
public class Circle {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  public Circle() {
    this(1.0);
  }

  public double getArea() {
    return this.radius * this.radius * Math.PI;
  }
}
```

this must be explicitly used to reference the data field radius of the object being constructed

this is used to invoke another constructor

Every instance variable belongs to an instance represented by this, which is normally omitted

Copy Constructor

```java
public class Circle {
      double radius;
      public Circle() {}
      public Circle(double radius) {
            this.radius = radius;
      }
      public Circle(Circle c) {
            this.radius = c.radius;
      }
}
```
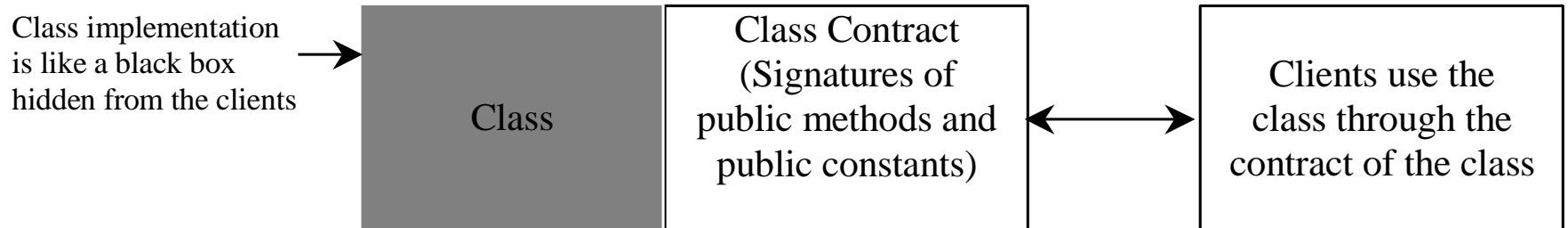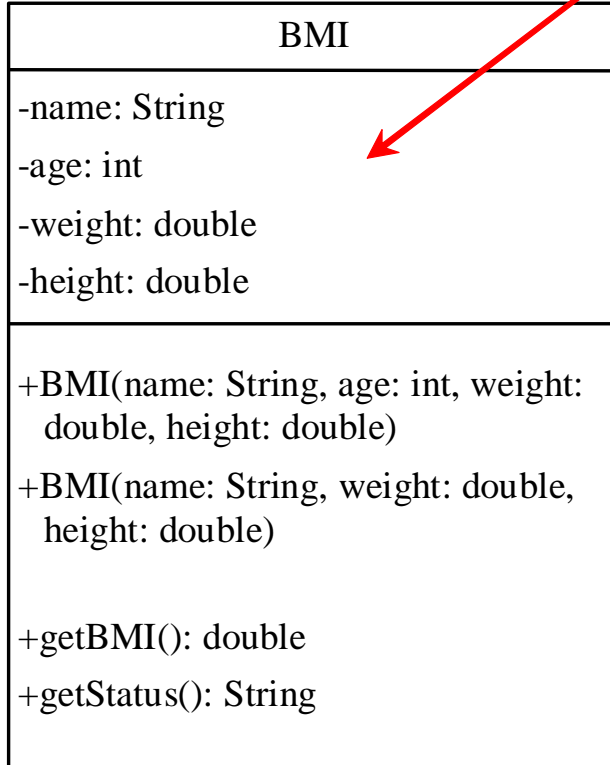
```java
// creating copy of an existing object
public static void main(String[] args) {
      Circle c1 = new Circle(5.0);
      Circle c2 = new Circle(c1);
}
```

Class Abstraction and Encapsulation

Class abstraction means to separate class implementation from the use of the class. The creator of the class provides a description of the class and let the user know how the class can be used. The user of the class does not need to know how the class is implemented. The detail of implementation is encapsulated and hidden from the user.

Class implementation is like a black box hidden from the clients →

Class

Class Contract (Signatures of public methods and public constants)	← →	Clients use the class through the contract of the class

The BMI Class

The get methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

BMI
-name: String
-age: int
-weight: double
-height: double
+BMI(name: String, age: int, weight: double, height: double)
+BMI(name: String, weight: double, height: double)
+getBMI(): double
+getStatus(): String

The name of the person.

The age of the person.

The weight of the person in pounds.

The height of the person in inches.

Creates a BMI object with the specified name, age, weight, and height.

Creates a BMI object with the specified name, weight, height, and a default age 20.

Returns the BMI

Returns the BMI status (e.g., normal, overweight, etc.)

BMI   UseBMIClass

Example: The Course Class

Course
-name: String
-students: String[]
-numberOfStudents: int
+Course(name: String)
+getName(): String
+addStudent(student: String): void
+getStudents(): String[]
+getNumberOfStudents(): int

The name of the course.

The students who take the course.

The number of students (default: 0).

Creates a Course with the specified name.

Returns the course name.

Adds a new student to the course list.

Returns the students for the course.

Returns the number of students for the course.

Course   TestCource   Run

Example: The `StackOfIntegers` Class

StackOfIntegers	
-elements: int[]	An array to store integers in the stack.
-size: int	The number of integers in the stack.
+StackOfIntegers()	Constructs an empty stack with a default capacity of 16.
+StackOfIntegers(capacity: int)	Constructs an empty stack with a specified capacity.
+empty(): boolean	Returns true if the stack is empty.
+peek(): int	Returns the integer at the top of the stack without removing it from the stack.
+push(value: int): int	Stores an integer into the top of the stack.
+pop(): int	Removes the integer at the top of the stack and returns it.
+getSize(): int	Returns the number of elements in the stack.

Designing the StackOfIntegers Class
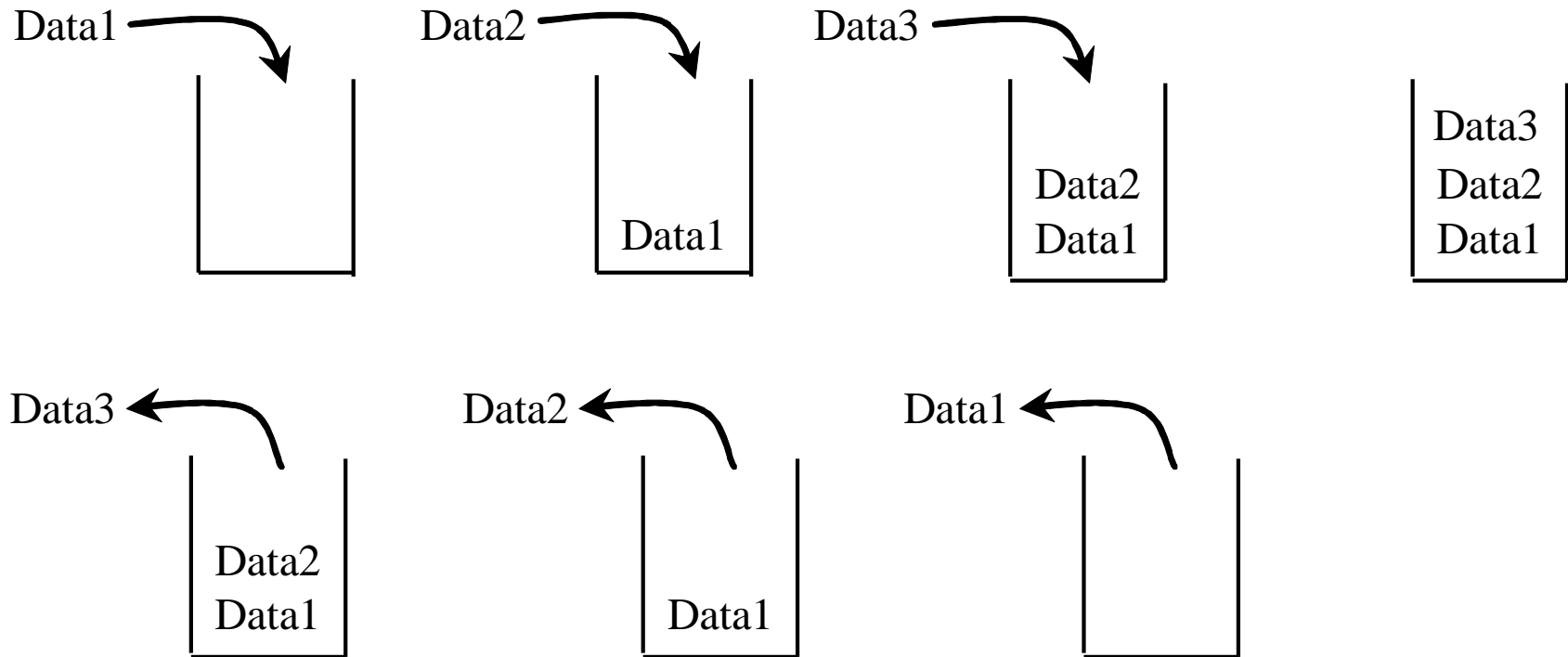
Implementing `StackOfIntegers` Class
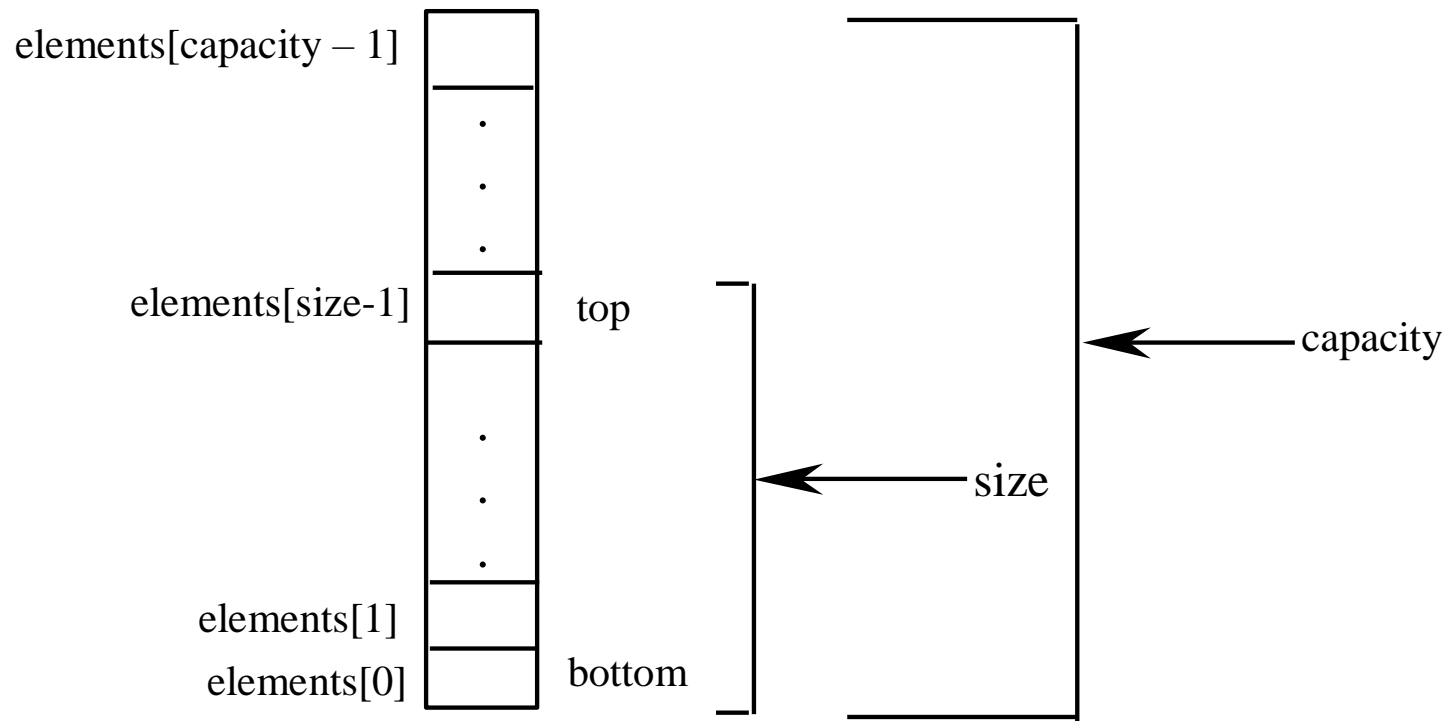
StackOfIntegers

Designing a Class

- (Coherence) A class should describe a single entity, and all the class operations should logically fit together to support a coherent purpose. You can use a class for students, for example, but you should not combine students and staff in the same class, because students and staff have different entities.

- (Separating responsibilities) A single entity with too many responsibilities can be broken into several classes to separate responsibilities. The classes String, StringBuilder, and StringBuffer all deal with strings, for example, but have different responsibilities. The String class deals with immutable strings, the StringBuilder class is for creating mutable strings, and the StringBuffer class is similar to StringBuilder except that StringBuffer contains synchronized methods for updating strings.

Designing a Class, cont.

- Classes are designed for reuse. Users can incorporate classes in many different combinations, orders, and environments. Therefore, you should design a class that imposes no restrictions on what or when the user can do with it, design the properties to ensure that the user can set properties in any order, with any combination of values, and design methods to function independently of their order of occurrence.

- Provide a public no-arg constructor and override the equals method and the toString method defined in the Object class whenever possible.

- Follow standard Java programming style and naming conventions. Choose informative names for classes, data fields, and methods. Always place the data declaration before the constructor, and place constructors before methods. Always provide a constructor and initialize variables to avoid programming errors.

Using Visibility Modifiers

- Each class can present two contracts – one for the users of the class and one for the extenders of the class. Make the fields private and accessor methods public if they are intended for the users of the class. Make the fields or method protected if they are intended for extenders of the class. The contract for the extenders encompasses the contract for the users. The extended class may increase the visibility of an instance method from protected to public, or change its implementation, but you should never change the implementation in a way that violates that contract.

- A class should use the private modifier to hide its data from direct access by clients. You can use get methods and set methods to provide users with access to the private data, but only to private data you want the user to see or to modify. A class should also hide methods not intended for client use.

Immutable Objects and Classes

If the contents of an object cannot be changed once the object is created, the object is called an *immutable object* and its class is called an *immutable class*. If you delete the set method in the <u>Circle</u> class in the preceding example, the class would be immutable because radius is private and cannot be changed without a set method.

A class with all private data fields and without mutators is not necessarily immutable. For example, the following class <u>Student</u> has all private data fields and no mutators, but it is mutable.

Example

```java
public class Student {
  private int id;
  private BirthDate birthDate;

  public Student(int ssn,
      int year, int month, int day) {
    id = ssn;
    birthDate = new BirthDate(year, month, day);
  }

  public int getId() {
    return id;
  }

  public BirthDate getBirthDate() {
    return birthDate;
  }
}
```

```java
public class BirthDate {
  private int year;
  private int month;
  private int day;

  public BirthDate(int newYear,
      int newMonth, int newDay) {
    year = newYear;
    month = newMonth;
    day = newDay;
  }

  public void setYear(int newYear) {
    year = newYear;
  }
}
```

```java
public class Test {
  public static void main(String[] args) {
    Student student = new Student(111223333, 1970, 5, 3);
    BirthDate date = student.getBirthDate();
    date.setYear(2010); // Now the student birth year is changed!
  }
}
```

What Class is Immutable?

For a class to be immutable, it must
(1) mark all data fields private and
(2) provide no mutator methods and
(3) no accessor methods that would return a reference to a
mutable data field object.

Multiple classes in a java program

```java
class Point {
        int x, y;
        public Point(int x, int y) {
                this.x = x; this.y = y;
        }
        public double distance(Point p) {
                double a = x - p.x;
                double b = y - p.y;
                return Math.sqrt(a*a-b*b);
        }
}

public class Circle {
        Point c;
        double r;
        public Circle(Point c, double r) {
                this.c = c;
                this.r = r;
        }
}
```

Circle.java

Point.class &
Circle.class

Inner Class

```java
public class Circle {
    class Point {
        int x, y;
        public Point(int x, int y) {
                this.x = x; this.y = y;
        }
        public double distance(Point p) {
                double a = x - p.x;
                double b = y - p.y;
                return Math.sqrt(a*a-b*b);
        }
    }


    Point c;
    double r;
    public Circle(int x, int y, double r) {
            this.c = new Point(x,y);
            this.r = r;
    }
}
```

Circle.java

Circle$Point.class
& Circle.class

Advantage of Inner Classes

C_Point.class
R_Point.class
Circle.class
Rectangle.class

Circle$Point.class
Rectangle$Point.class
Circle.class
Rectangle.class

Inner Class