Chapter 9 Thinking in Objects

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.

```java
public class Circle {
    private double r;
    public Circle(double radius) {
        r = radius;
    }
    public double getRadius() {
        return r*r*Math.PI;
    }
}
```

- A class with all private data fields and without mutators is not necessarily immutable.

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;
    }
}
```

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.

What Class is Immutable?
- For a class to be immutable, it must mark all data fields private and provide no mutator methods and no accessor methods that would return a reference to a mutable data field object.

Scope of Variables
- The scope of instance and static variables is the entire class. They can be declared anywhere inside a class.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be initialized explicitly before it can be used.
Object-Oriented Thinking

- Classes provide more flexibility and modularity for building reusable software

### The BMI Class

- **name**: String
- **age**: int
- **weight**: double
- **height**: double

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

- **name**: The name of the person.
- **age**: The age of the person.
- **weight**: The weight of the person in pounds.
- **height**: The height of the person in inches.

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

### Example: The Course Class

- **name**: String
- **students**: String[]
- **numberOfStudents**: int

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

- **name**: The name of the course.
- **students**: The students who take the course.
- **numberOfStudents**: The number of students (default: 0).

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

### Designing the StackOfIntegers Class

- **elements**: int[]
- **size**: int

- `StackOfIntegers()`
  - `getSize()`: int

- `StackOfIntegers(int capacity)`
  - `push(int value)`: int
  - `pop()`: int

- `empty()`: boolean
- `peek()`: int

- **elements**: An array to store integers in the stack.
- **size**: The number of integers in the stack.

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

### Implementing StackOfIntegers Class

- Stack of integers
  - `top`: int
  - `capacity`: int

- `push(int value)`: void
  - `pop()`: int
  - `getSize()`: int

- **elements**: An array to store integers in the stack.
- **size**: The number of integers in the stack.

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