NYU

Courant Institute of Mathematical Sciences
Department of Computer Science
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

Chapter#10: Working with Objects and Classes

Objectives

- Introducing Immutable objects and classes
- Learning the concept of Variable Scope, the this references
- Introducing Abstraction and Encapsulation and Object Composition
- Designing a Stack Class
- Learning how to design a class and designing wrapper class for primitive data types
- Introducing *BigInteger* and *BigDecimal* Classes

Immutable Classes and Objects

Immutable objects are objects whose values cannot be changed

Requirements:

- All Data fields must be private
- No Getters that will return a reference to a data field
- No Setters that will modify the data fields

Scope of Variables

Data fields are variables of a class and their scope covers the entire class. As such, they possess *class scope*.

Local variables are variables declared and defined within methods. As such their scope is *local*. They can only be accessed from within the method.

Data fields and methods can be declared in any order. However, one should declare the data fields together before the methods.

Local variables whose names are the same as the class's variables hide the class' variable within that method. Hidden variables are variables with a hidden class scope.

"this" Reference

"this" refers to the object itself. Within the class definition you can replace any mention of a data field or a method by preceding it with *"this"*

There are two key reasons for why we need it:

- To access hidden data fields
- To invoke the constructor from another constructor

Class Abstraction and Encapsulation

Class abstraction separates the implementation of a class from its use.

Class contract is the collection of methods and fields that are accessible from outside of the class with descriptions of how these members are expected to behave.

Class encapsulation means that the class implementation is hidden from the user of the class.

Abstract data type is another name for a class whose implementation is hidden.

Object composition

An object can contain another object. This relationship is called a **has-a** relationship.

For example, a **Student** object has-a:

- name, a String object
- date of birth, a Date object
- address, an Address object

The "owner" object is called the **aggregating object** and it belongs to the **aggregating class**.

The "subject" object is called an **aggregated object** and it belongs to the **aggregated class.**

Designing a **Stack** Class

A stack is a data structure that stores data in a last-in, first-out (LIFO) order where new items are added to the top and older items are at the bottom. Items in this structure can only be removed from the top of the stack.

StackOfCharacters class contract:

StackOfCharacters() – Constructs an empty stack of characters with a default capacity of 16.

StackOfCharacters(capacity: int) – Constructs an empty stack of characters with a specified capacity (if provided capacity is less than or equal to zero, the capacity is set to default 16).

Empty(): Boolean – Returns true if the stack is empty, otherwise it returns false.

Designing a **Stack** Class

getSize(); int - Returns the number of elements in the stack.

getCapacity(): int – Returns the capacity of the stack.

push (value: Character): void – Puts the character stack at the top of the stack.

pop(): Character – Removes the character from the top of the stack and returns it. If the stack is empty, returns null.

peek(): Character - Returns the character from the top of the stack (without removing it). If stack is empty, returns null.

Class Design Guidelines

Cohesion

The class should describe a single thing to maintain consistency with Java programming conventions.

Place data fields before constructors and constructors before other methods.
Provide default (no argument) constructors.

Use **standard** methods and field names
Example: length, **compareTo**, **toString**.

Implement the **toString** method.
Implement the **compareTo** method.

Class Design Guidelines

Encapsulation

- Refers to the practice of hiding implementation details
- Make all data fields private
  - Provide get methods for fields that should be readable
  - Provide set methods for fields that should be writable
  - Make helper methods private (e.g getters and setters)
    - Helper methods are the methods that should not be called from outside of the class

Class Design Guidelines

Clarity

Methods should implement simple tasks.

Methods should be independent

They should not rely on one another to start/exit. Provide a graceful way of quitting the method rather than letting it crash the program.

Use independent data fields

Do not keep multiple data fields that can be derived one from another. Having data fields that are not independent implies that they all need to be modified when one of them changes.

Exception to the rule: Cases where computing a value is costly.

Class Design Guidelines

- Completeness
  - Provide both a full and general functionality for the class.
    - Ensure that your class is functional but allow it to be used for more than one purpose.
    - Think of all the different ways in which a class can be used.

Wrapper Classes for Primitive Data Types

Java provides class wrappers for all primitive data types

- Character
- Byte
- Short
- Integer
- Long
- Float
- Double
- Boolean

Wrapper Classes for Primitive Data Types

A primitive type value can be automatically converted to an object using a wrapper class, and vice versa, depending on the context.

Familiarize yourself with the methods provided in these classes. They will prove to be useful in many situations.

Wrapper Classes for Primitive Data Types

Integer intObject = new Integer(5);

Is equivalent to

Integer intObject = 5;

You can also write:

```
Integer intObject = 5;
Int primitiveInt = intObject;
```

Converting a primitive value to a wrapper object is called **boxing**. The reverse conversion is called **unboxing**.

BigInteger and BigDecimal Classes

From the Oracle Docs:

BigInteger

> Provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and others.

BigDecimal

> Provides operations for arithmetic, scale manipulation, rounding, comparison, hashing, and format conversion
>
> For more details: https://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html

NYU

Courant Institute of Mathematical Sciences
Department of Computer Science
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

Chapter#10: Working with Objects and Classes