# CS3500: Object-Oriented Design
# Spring 2014

## Class 3
## 1.14.2014



Northeastern University
College *of* Computer and Information Science

# Assignment 1

- Due: Tuesday, January 14, 2014 (TONIGHT) at 11:59 pm
- Recipe implementation in Java of the MySet ADT
- MySet is an immutable abstract data type whose values represent finite sets with elements of type Long.

Why would this code receive the hint

`hashCode {4,3,2,1} {3,4,2,1}`?

```
public int hashcode() {
    return MySet.size(this);
}
```

```
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

/**
 * Testing class Y
 *
 * @author name
 * @version date/version
 */
public class YTest {
    //TODO: declare fields

    /**
     * setUp for the tests
     */
    @Before
    public void setUp() {
        //TODO: initialize fields
    }

    /**
     * testing method x
     */
    @Test
    public void testX() {
        //TODO: add tests for method x. These tests will
        //include the use of assertTrue, assertFalse, etc.
    }
}
```

# Review Definitions with StackInt

# Abstract Class vs. Concrete Class

- Concrete class: full implementation of the type.
- Abstract classes: at most a partial implementation of the type

Signature:

Public static methods (of the Abc class):

```
defg     : Abc x int  -->  int
hijk     : Abc x int  -->  Abc
lmno     : Abc x int  -->  Abc
pqrs     : int        -->  Abc
tuvw     : Abc        -->  int
```

Algebraic Specification:

```
Abc.defg (Abc.lmno (u, k), n)
    =  Abc.defg (u, n)                          if n < Abc.tuvw (u)
Abc.defg (Abc.lmno (u, k), n)
    =  k                                        if n == Abc.tuvw (u)
Abc.defg (Abc.lmno (u, k), n)
    =  n                                        if n > Abc.tuvw (u)
Abc.defg (Abc.pqrs (k), n)
    =  3
Abc.hijk (Abc.lmno (u, k), n)
    =  Abc.lmno (Abc.hijk (u, n), k)            if n < Abc.tuvw (u)
Abc.hijk (Abc.lmno (u, k), n)
    =  Abc.lmno (u, n + 1)                      if n == Abc.tuvw (u)
Abc.hijk (Abc.lmno (u, k), n)
    =  u                                        if n > Abc.tuvw (u)
Abc.hijk (Abc.pqrs (k), n)
    =  Abc.lmno (Abc.pqrs (0), k)
Abc.tuvw (Abc.lmno (u, k))
    =  1 + Abc.tuvw (u)
Abc.tuvw (Abc.pqrs (k))
    =  0
```

# Abstract Data Type (ADT)

- What is an ADT?
  - set of data
  - set of operations
  - description of what operations do
- Within this course, when discuss ADTs, we will discuss them using:
  - a signature: names of operations and types
  - a specification: agreement between client and implementors

# Abstraction Barrier

- Every piece of software has, or should have, an abstraction barrier that divides the world into two parts: clients and implementors.
  - The clients are those who use the software. They do not need to know how the software works.
  - The implementors are those who build it. They need to know how the software works.

# Abstraction Barrier

Client

- Knows the behavior of the data type
- Doesn't know how the data type was implemented, but can use the data type based on the specs

Abstraction Barrier

Implementor

- Knows the behavior of the data type
- Knows how the data type was implemented

# Interchangeable Parts
## [history.com]

- 18th century: Guns were made by hand, which meant each gun was one-of-a-kind. To repair a gun, a part had to be made especially for it.
- Mid-18th century a French gunsmith Honoré LeBlanc suggested the gun parts be made from standardized patterns, so that all gun parts would follow the same design and could be easily replaced if broken.
- Early 19th century, Eli Whitney manufacturing muskets—production of large numbers of identical parts quickly and at a relatively low cost.
- The U.S. introduced the first large-scale assembly of weapons with its adoption of the Model 1842 musket, and the new arms industry would produce hundreds of thousands of rifles for Civil War soldiers, all from interchangeable parts.

Image from: http://www.cesura17.net/~will/Pictures/Places/Images/2012csc3.jpg

# Liskov Chapter 2: Review of Objects in Java

# Packages and Visibility

- `private` - accessible only within the same class
- (default) - accessible only within the same package
- `protected` - accessible within the same package and also accessible within subclasses
- `public` - accessible everywhere

# Type Checking

- Type safety
- Strongly type checking

# Type Hierarchy
## [Liskov, p.27, Sidebar 2.4]

- Java supports *type hierarchy*, in which one type can be the *supertype* of other types, which are its *subtypes*. A subtype's objects have all the methods defined by the supertype.
- All object types are subtypes of `Object`, which is the top of the type hierarchy. `Object` defines a number of methods, including `equals` and `toString`. Every object is guaranteed to have these methods.
- The *apparent type* of a variable is the type understood by the compiler from information available in declarations. The *actual type* of an object is its real type—the type it receives when it is created.
- Java guarantees that the apparent type of an expression is a super type of its actual type.

# Types

# Stream Input/Output

```
System.in  //Standard input to the program.

System.out //Standard output from the program.

System.err //Error output from the program.


//Read from console
Scanner sc = new Scanner(System.in);
```

# Liskov Chapter 1: Introduction

# Decomposition & Abstraction

[Liskov]

- Decomposition: A way of dividing a large problem into smaller subproblems. We do decomposition through the recognition of useful abstractions.

- Abstraction: The process of forgetting information so that things that are different can be treated as if they are the same. Also, the program entity that results from the abstraction process.

# Abstraction Mechanisms

[Liskov]

- Abstraction by parameterization
- Abstraction by specification

# Kinds of Abstractions

[Liskov]

- Procedural abstraction
- Data abstraction
- Iteration abstraction
- Type hierarchy