CS206 Data Structures

Brief Intro. to JAVA

Sung-eui Yoon (윤성의)

Department of Computer Science
KAIST

http://sglab.kaist.ac.kr/~sungeui

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Class Objectives

- ☐ Understand basic concepts of JAVA
- ☐ Walk through the programming procedure
- ☐ Use Junit for testing your homework
- ☐ Be ready for Programming Assignment 0

- ☐ These are from Chap. 1 and 2

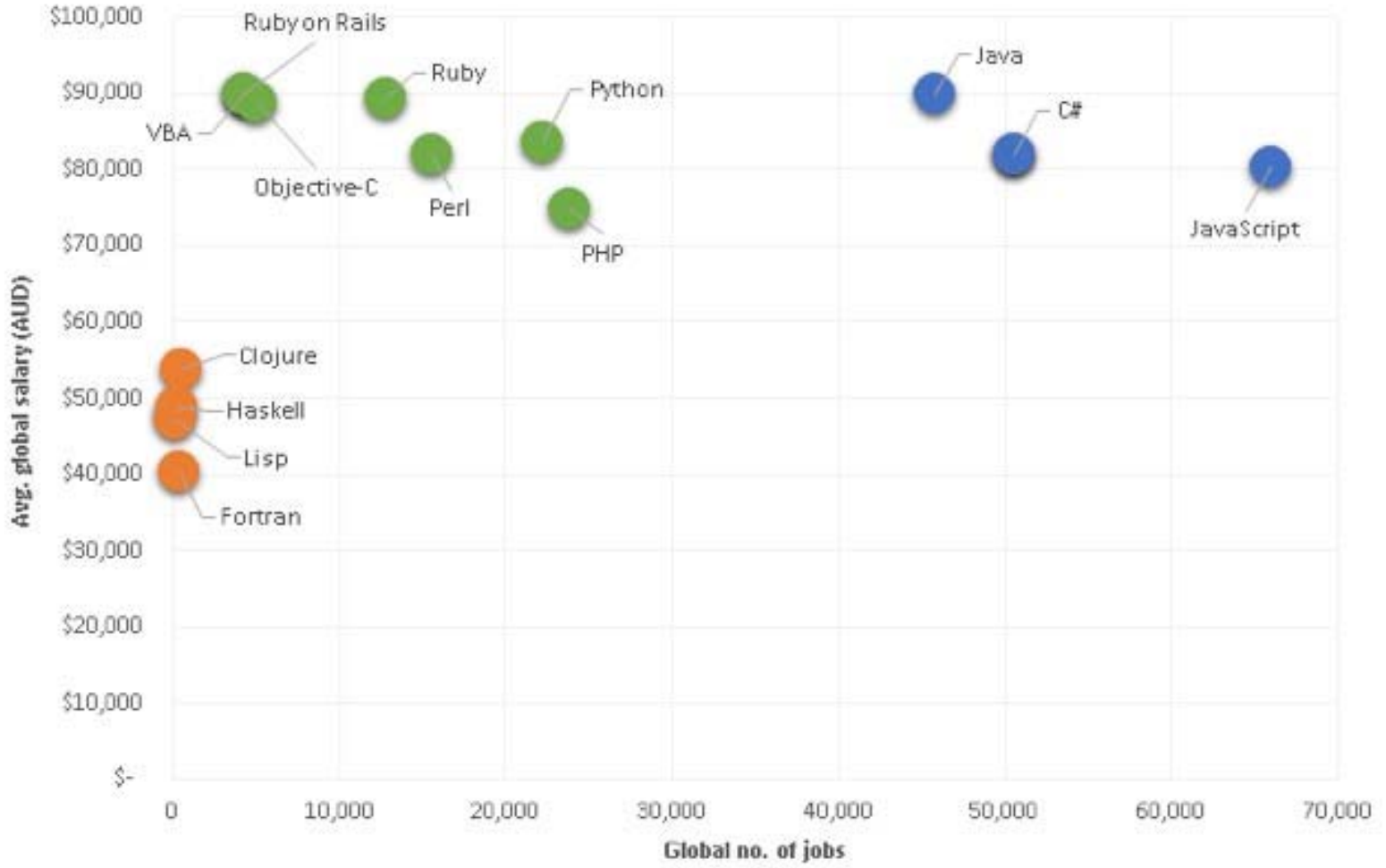
The Java Programming Language

Position Jul 2010	Position Jul 2009	Programming Language	Ratings Jul 2010	Delta Jul 2009	Status
1	1	Java	18.673%	-1.78%	A
2	2	C	18.480%	+1.16%	A
3	3	C++	10.469%	+0.05%	A
4	4	PHP	8.566%	-0.70%	A
5	6	C#	5.730%	+1.19%	A
6	5	(Visual) Basic	5.516%	-2.27%	A
7	7	Python	4.217%	-0.22%	A
8	8	Perl	3.099%	-1.10%	A
9	21	Objective-C	2.498%	+1.99%	A
10	9	JavaScript	2.432%	-1.08%	A
11	11	Delphi	2.323%	+0.33%	A
12	10	Ruby	1.982%	-0.59%	A
13	12	PL/SQL	0.772%	-0.12%	A
14	13	SAS	0.701%	-0.09%	A
15	15	Pascal	0.639%	-0.07%	A--
16	17	Lisp/Scheme/Clojure	0.622%	+0.01%	B
17	20	MATLAB	0.581%	+0.07%	B
18	16	ABAP	0.548%	-0.15%	B
19	19	Lua	0.535%	+0.00%	B
20	28	PowerShell	0.493%	+0.17%	B

Data from www.tiobe.com

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

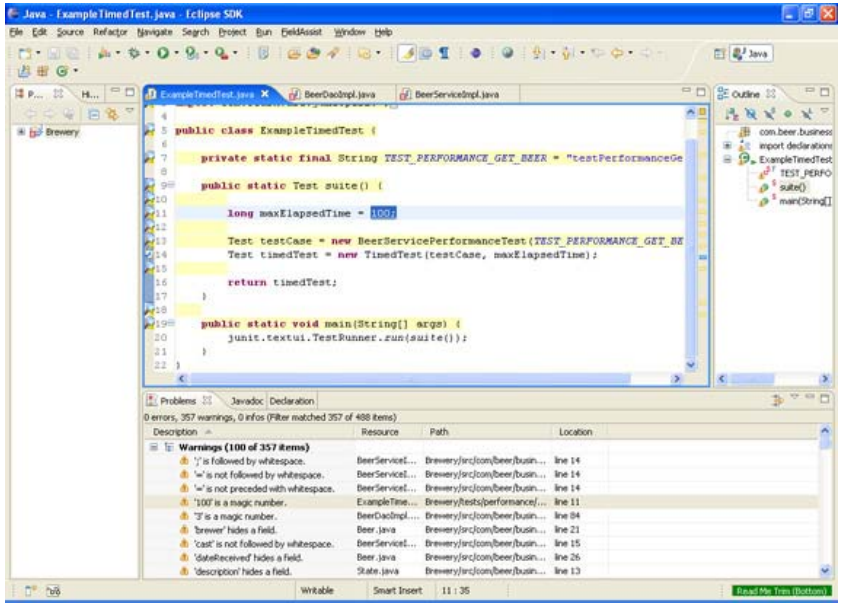
Language in terms of Salary and Demand

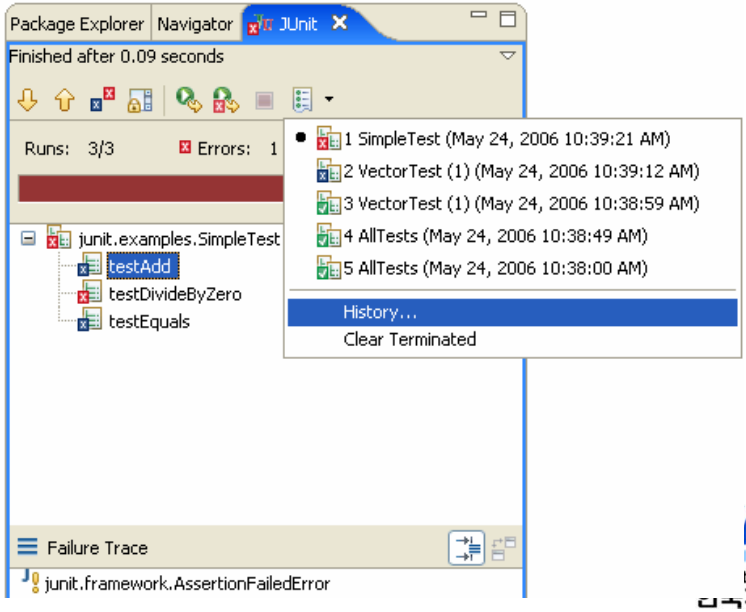


KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Essential Java Programming Tools

- ☐ Eclipse (IDE; Integrated Development Environment)
  - Download from www.eclipse.org
- ☐ Java Programming Tutorial - 3 - Downloading Eclipse
  - http://www.youtube.com/playlist?list=PLFE2CE09D83EE3E28
  - Local copies are at http://sglab.kaist.ac.kr/~sungeui/DS/video_tutorial/
- ☐ JUnit (included in eclipse)





AIST
Institute of Science and Technology
한국과학기술원

Basic JAVA Syntax

- Similar in some extent to Python and C++
- Learning JAVA language itself is not the goal of this course
  - Need to know basic JAVA that are needed for finishing homeworks
  - Focus on data structures and operations with them

Inheritance

☐ Want: compute progression

- Numeric progression: $x_{t+1} = x_t + 1$
- Arithmetic progression: $x_{t+1} = x_t + f$
- Geometric progression: $x_{t+1} = x_t * f$
- Fibonacci progression: $x_{t+1} = x_t + x_{t-1}$

☐ Would you write separate program for each progression?

☐ Let's try object-oriented design using inheritance

- Need first value (x_0) and current value (x_t) to compute next value (x_{t+1})
- Progression classes will have long-integer fields
  - `first`
  - `cur`

Numeric Progression

```
/**
 * A class for numeric progressions.
 */
public class Progression
{

    /** First value of the progression. */
    protected long first;

    /** Current value of the progression */
    protected long cur;

    /** Default constructor. */
    public Progression()
    {
        cur = first = 0;
    }

    /**
     * Resets the progression to the first
value.
     *
     * @return first value
     */
    public long firstValue()
    {
        cur = first;
        return cur;
    }
```

```
    /**
     * Advances the progression to the
next value.
     *
     * @return next value of the
progression
     */
    public long nextValue()
    {
        return ++cur; // default next
value
    }

    /**
     * Prints the first n values of the
progression.
     *
     * @param n
     *             number of values to
print
     */
    public void printProgression(int n)
    {
        System.out.print(firstValue());
        for (int i = 2; i <= n; i++)
            System.out.print(" " +
nextValue());
        System.out.println(); // ends the
line
    }
}
```

Commonly used JAVA keywords

☐ **Base types**

- **int**: 4 bytes integer
- **long**: 8 bytes integer; we can represent a big integer that can be represented within 8 bytes

☐ **Modifiers**

- **public**: Anyone can access it
- **protected**: Only methods of the same package or of its subclass can access
- **private**: Only methods within the same class can access. The default modifier is friendly, which means that any class in the same package can access.
- **static**: The variable is associated with the class, not the instance of the class. This serves as a kind of global variables
- **final**: A final variable plays as a constant value

☐ **this**: this refers to the current instance of that class

Creating and Using Objects

```
/** Test program for the progression classes */
class TestProgression
{
    public static void main(String[] args)
    {
        Progression prog;

        // test ArithProgression
        System.out.println("Arithmetic progression with default increment:");
        prog = new ArithProgression();
        prog.printProgression(10);
    }
}
```

- We declare objects first and create objects (or instances) by calling **new**
  - During creating objects, we call a constructor
  - We access methods and instance variables by using the dot operator

Arithmetic Progression

```
/**
 * Arithmetic progression.
 */
public class ArithProgression extends Progression
{

    /** Increment. */
    protected long inc;

    // Inherits variables first and cur.

    /** Default constructor setting a unit increment. */
    public ArithProgression()
    {
        this(1);
    }

    /** Parametric constructor providing the increment. */
    public ArithProgression(long increment)
    {
        inc = increment;
    }
    /** Advances the progression by adding the increment to the current value.
     *
     * @return next value of the progression
     */
    public long nextValue()
    {
        cur += inc;
        return cur;
    }

    //  Inherits methods firstValue() and printProgression(int).
}
```

Geometric Progression

```
/**
 * Geometric Progression
 */
class GeomProgression extends Progression
{
    /** Base. */
    protected long base;

    // Inherits variables first and cur.

    /** Default constructor setting base 2. */
    public GeomProgression()
    {
        this(2);
    }

    /** Parametric constructor providing the base.
     *
     * @param b base of the progression.
     */
    public GeomProgression(long b)
    {
        base = b;
        first = 1;
        cur = first;
    }

    /** Advances the progression by multiplying the base with the current value.
     *
     * @return next value of the progression
     */
    public long nextValue()
    {
        cur *= base;
        return cur;
    }

    //  Inherits methods firstValue() and printProgression(int).
}
```

Fibonacci Progression

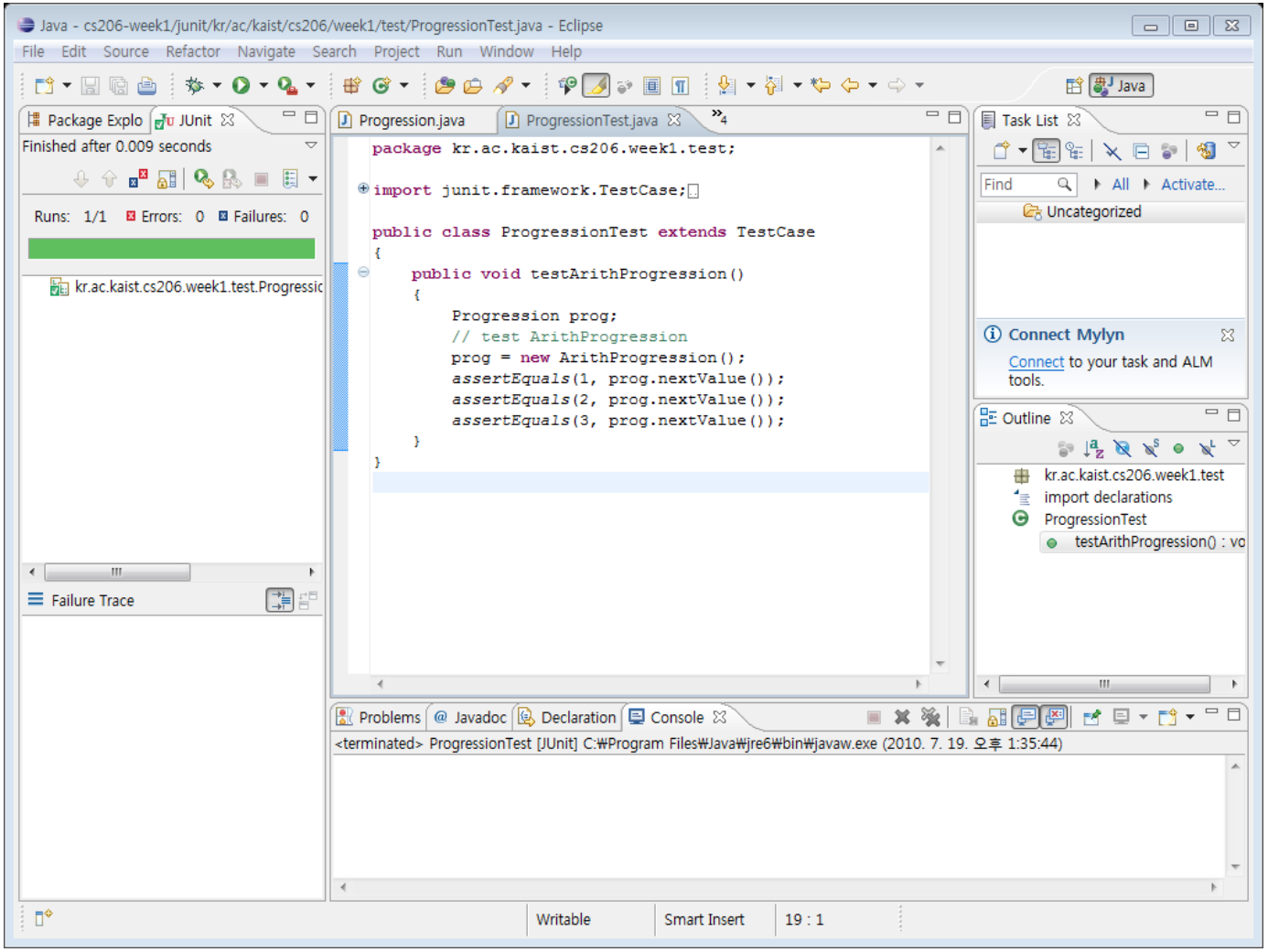
```
/**
 * Fibonacci progression.
 */
class FibonacciProgression extends Progression
{
    /** Previous value. */
    long prev;

    // Inherits variables first and cur.

    /** Default constructor setting 0 and 1 as the first two values. */
    FibonacciProgression()
    {
        this(0, 1);
    }

    /** Parametric constructor providing the first and second values.
     *
     * @param value1 first value.
     * @param value2 second value.
     */
    FibonacciProgression(long value1, long value2)
    {
        first = value1;
        prev = value2 - value1; // fictitious value preceding the first
    }

    /** Advances the progression by adding the previous value to the current value.
     *
     * @return next value of the progression
     */
    public long nextValue()
    {
        long temp = prev;
        prev = cur;
        cur += temp;
        return cur;
    }
    // Inherits methods firstValue() and printProgression(int).
}
```

Testing Progression

```
/** Test program for the progression classes */
class TestProgression
{
    public static void main(String[] args)
    {
        Progression prog;
        // test ArithProgression
        System.out.println("Arithmetic progression with default increment:");
        prog = new ArithProgression();
        prog.printProgression(10);
        System.out.println("Arithmetic progression with increment 5:");
        prog = new ArithProgression(5);
        prog.printProgression(10);
        // test GeomProgression
        System.out.println("Geometric progression with default base:");
        prog = new GeomProgression();
        prog.printProgression(10);
        System.out.println("Geometric progression with base 3:");
        prog = new GeomProgression(3);
        prog.printProgression(10);
        // test FibonacciProgression
        System.out.println("Fibonacci progression with default start values:");
        prog = new FibonacciProgression();
        prog.printProgression(10);
        System.out.println("Fibonacci progression with start values 4 and 6:");
        prog = new FibonacciProgression(4, 6);
        prog.printProgression(10);
    }
}
```

Trying Out in Eclipse

- I will show you how to use JUnit instead of the `TestProgression` class



Testing Progression using Junit

```
package testprog;

import static org.junit.Assert.*;    // import Junit packages
import org.junit.Test;

public class ProgressionTest {

        @Test              // Junit test functions with @Test
        public void test() {
                Progression prog;

                prog = new ArithProgression ();
                assertEquals (1, prog.nextValue ()); // raise error if it is not same
                assertEquals (2, prog.nextValue ());
                assertEquals (3, prog.nextValue ());
        }

}
```

Exceptions

☐ Objects thrown by code when unexpected condition happens

- Extends `Exception` class

☐ Throwing an exception

```
if (insertIndex >= A.length) {
    throw new
    BoundaryViolationException("No element at index " + insertIndex);
}
```

```
    public void goShopping() throws ShoppingListTooSmallException,
                                    OutOfMoneyException
    {    // method body . . .
    }

    public void getReadyForClass() throws ShoppingListTooSmallException,
                                          OutOfMoneyException
    {
        goShopping(); // I don't have to try or catch the exceptions
                      // which goShopping() might throw because
                      // getReadyForClass() will just pass these along.
        makeCookiesForTA();
    }
```

Catching Exceptions

☐ Uncaught exceptions…

```
java.lang.NullPointerException: Returned a null locator
   at java.awt.Component.handleEvent(Component.java:900)
   at java.awt.Component.postEvent(Component.java:838)
   at java.awt.Component.postEvent(Component.java:845)
   at sun.awt.motif.MButtonPeer.action(MButtonPeer.java:39)
   at java.lang.Thread.run(Thread.java)
```

☐ Catching exceptions

```
int index = Integer.MAX_VALUE; // 2.14 Billion
try
// This code might have a problem...
{
    String toBuy = shoppingList[index];
} catch (ArrayIndexOutOfBoundsException aioobx)
{
    System.out.println("The index "
            + index
            + " is outside the array.");
}
```

Interfaces

- Users just need to know methods that each object can support
  - E.g., turn the wheel left, if you want to move the car left
  - Users don't need to know how it works

- They are represented in:
  - Application Programming Interface (API) or
  - Interface

- A collection of method declarations with no data and no bodies

Interfaces

```
/** Interface for objects that can be sold. */
public interface Sellable
{
    /** description of the object */
    public String description();
    /** list price in cents */
    public int listPrice();
    /** lowest price in cents we will accept */
    public int lowestPrice();
}

/** Class for photographs that can be sold */
public class Photograph implements Sellable
{
    private String descript; // description of this photo
    private int price; // the price we are setting
    private boolean color; // true if photo is in color

    public Photograph(String desc, int p,
            boolean c) // constructor
    { descript = desc; price = p; color = c; }
    public String description()
    {       return descript;    }
    public int listPrice()
    {       return price;    }
    public int lowestPrice()
    {       return price / 2;    }
    public boolean isColor()
    {       return color;    }
}
```

Multiple Interfaces

```
/** Interface for objects that can be
transported. */
public interface Transportable
{
    /** weight in grams */
    public int weight();

    /** whether the object is hazardous */
    public boolean isHazardous();
}

/** Class for objects that can be sold,
packed, and shipped. */
public class BoxedItem implements Sellable,
        Transportable
{

    private String descript; //
description of this item
    private int price; // list price in
cents
    private int weight; // weight in grams
    private boolean haz; // true if object
is hazardous
    private int height = 0; // box height
in centimeters
    private int width = 0; // box width in
centimeters
    private int depth = 0; // box depth in
centimeters
```

```
    /** Constructor */
    public BoxedItem(String desc, int p,
int w,
            boolean h)
    {
        descript = desc; price = p; weight
= w; haz = h;
    }
    public String description()
    {       return descript;    }
    public int listPrice()
    {       return price;    }
    public int lowestPrice()
    {       return price / 2;    }
    public int weight()
    {       return weight;    }
    public boolean isHazardous()
    {       return haz;    }
    public int insuredValue()
    {       return price * 2;    }
    public void setBox(int h, int w, int d)
    {
        height = h; width = w;
        depth = d;
    }
}
```

Multiple Inheritance in Interfaces

☐ An interface can extend more than one interfaces

```
public interface InsurableItem extends
        Transportable, Sellable
{
    /** Returns insured Value in cents */
    public int insuredValue();
}

public class BoxedItem2 implements InsurableItem {

  // ... same code as class BoxedItem
}
```

- What about classes?

Class Objectives were:

- ☐ Understand basic concepts of JAVA
- ☐ Walk through the programming procedure
- ☐ Use JUnit for testing your homework

- ☐ These are from Chap. 1 and 2

Any Questions? (Question HW)

☐ Come up with one question on what we have discussed in the class and submit at the end of the class

- 1 for typical questions (that have been already answered in the class)
- 2 for questions with thoughts or or that surprised me

☐ Write questions at least 4 times

- Write a question about one per month
- Anything related to class materials is okay
- Unrelated questions are not counted (e.g., faculty club에 학부생도 출입이 가능한가요?)

☐ You can type online (KLMS)

My Responses to Those Questions

- ☐ Identify common questions and address them at the Q&A file
- ☐ Some of questions will be discussed in the class

- ☐ If you want to know the answer of your question, ask me or TA on person
  - Feel free to ask questions in the class

- ☐ We are focusing on having good questions!
  - All of us are already well trained for answering questions

Programming Assignment 0

☐ Create a program of “Hello World”

- Do it by yourself
- Submit it (codes and binary) by 11:59pm, Sep.-11 (Thur.)

☐ Todo:

- Download the eclipse
- Just print “Hello JAVA”
- Compile and run it, and play with the eclipse

☐ JAVA online lecture materials

- Java Programming Tutorial - 4 - Hello YouTube
- http://www.youtube.com/playlist?list=PLFE2CE09D83EE3E28

☐ JAVA materials

- Tutorial: http://docs.oracle.com/javase/tutorial/
- Reference: http://docs.oracle.com/javase/7/docs/api/

Lab.

☐ 9/15 (Mon.)

- Starts at 7:00pm, N1-201
- 3 TAs will be there
- 50 computers are there; first come first served

☐ If you can follow online video well, you don't need to attend the lab.

☐ Otherwise, please attend it and get supports from TAs

Homework

- ☐ Go over the next lecture slides before the class
  - Just 10 min ~ 20 min should be okay

Next Time

☐ Array and linked list