Final Review

Cheng, Wei  COMP110-001  June 13, 2014
int a = 10;
int b = a;
b = b + 1;

System.out.println(a);
Student anna = new Student();
anna.PID = 1234;
anna.year = 3;

Student a_copy = anna;
a_copy.year = 4;

System.out.println( anna.year );
Pass-by-Value

- When a method with parameter of primitive type is called?

```java
public void increaseByOne( int num ) {
    num = num + 1;
}

public void doSth () {
    int someNum = -2;
    increaseByOne( someNum );
    System.out.println( someNum );
}
```

What do you get?
Pass-by-Value

• When a method with parameter of Class type is called?

```java
public void increaseByOne( Student s) {
    s.year = s.year + 1;
}

public void doSth () {
    Student anna = new Student();
    anna.PID = 1234;
    anna.year = 3;
    increaseByOne( anna );
    System.out.println( anna.year );
}
```

What do you get?
Information Hiding

• **Software:**
  – usually efforts of many engineers
  – Divided into multiple components
  – Each component interacts with other components
  – Each component has its internal data/logic that are not supposed to be visible to outside
  – We will see examples later
Design a method so that it can be used without any need to understand the fine detail of the code is called information hiding.

```java
/**
 * Precondition: The instance variables of the calling object have values.
 * Postcondition: The data stored in (the instance variables of) the receiving object have been written to the screen.
 */
public void writeOutput()

/**
 * Precondition: years is a nonnegative number.
 * Postcondition: Returns the projected population of the receiving object after the specified number of years.
 */
public int predictPopulation(int years)
```
Access Control Modifiers

**public**: attributes/methods that can be used (invoked) by any other classes without restriction

--- object interaction is done through these attributes/methods

protected: not covered in this course

default (no modifier): covered later

**private**: attributes/methods that is only available within the class (i.e., cannot be invoked from outside)
Constructors

• Constructor is a special method that is called when a new object is created

Student berkeley; // not called

Student berkeley = new Student();
  // called with new keyword
Let’s write our first constructor:

```java
public class Student {
    private int PID;
    private int year;
    .... Accessors & mutators ..... 

    public Student( int PID, int year ) {
        this.PID = PID;
        this.year = year;
    }
}
```

There is no return type or “void” keyword

Constructor has the same name as the class
Constructors

public Student( int PID, int year ) {
    this.PID = PID;
    this.year = year;
}

• With this constructor, we can now do:
  Student berkeley = new Student( 1234, 2 );
Multiple Constructors

- You can have multiple constructors in one class. They all have the same name, just different parameters.

```java
public class Student {
    ...
    
    public Student( int PID, int year ) {
        this.PID = PID;
        this.year = year;
    }

    public Student( int PID ) {
        this.PID = PID;
        this.year = 1; // default case – the 1st year
    }
}
```
Constructors

• Generally, constructor should contain all initialization logic
  – assign initial values based on input parameters
  – assign default initial values without input
  – reserve resource, prepare input/output stream
  – whatever other logic necessary (e.g., error checking)

• We will see more examples later.
Default Constructor

• What if you did not write any constructor?

```java
public class Student {
    private int PID;
    private int year;
    .... No constructor .....
}

Student berkeley = new Student();
```

Java gives each class a default constructor **if you did not write any constructor**. It assigns a default value to each instance variable.

- integer, double: 0
- String and other class-type variables: null
- boolean: false
Constructors

• If you define at least one constructor, a default constructor will not be created for you
Static Members

• static variables and methods belong to a class as a whole, not to an individual object

Sounds weird, doesn’t it?
  - static is against OO in some sense

• Where have we seen static before?
• When would you want a method that does not need an object to be called?
• Static variables and methods can be accessed using the class name itself:
  – `System.out.println( Math.Pi );`
  – `int z = Math.pow(2, 4);`
static vs non-static

• All static members are at class level. They are accessed without creating any instance.

• Thus, there is no “current object” in writing static methods.

• static methods has no access to instance variables or non-static methods (since they belong to instances)
Creating an array

```java
int[] scores = new int[5];
```

• This is like declaring 5 strangely named variables of type `int`:
  – `scores[0]`
  – `scores[1]`
  – `scores[2]`
  – `scores[3]`
  – `scores[4]`
Array details

• Syntax for creating an array:

  \texttt{Base\_Type[] \texttt{Array\_Name} = new \texttt{Base\_Type[Length]}}

• Example:

  \texttt{int[] \texttt{pressure} = new \texttt{int[100];}}

• Alternatively:

  \texttt{int[] \texttt{pressure;}}
  \texttt{ pressure = new \texttt{int[100];}}
Finding the length of an existing array

• An array is a special kind of object
  – It has one public instance variable: `length`
  – `length` is equal to the length of the array
    
    ```java
    Pet[] pets = new Pet[20];
    pets.length has the value 20
    ```
  – You cannot change the value of `length`
  – *Once declared, an array cannot be resized!*
    
    • *Why array cannot be resized?*
“Dynamic Array”

This is a common problem and the solution is quite complicated.

Java has several built-in classes that implements the copy-once-get-full strategy and can serve as a “dynamic array” – array that can be resized.

A popular one is ArrayList.
ArrayList

- Internally, it maintains an array of specified type.
- You can view it as a list of data.
- To initialize a list of particular type:

```java
ArrayList<Data_type> var = new ArrayList<Data_type>();
```

  e.g.: `ArrayList<Student> myList = new ArrayList<Student>();`

  or

```java
ArrayList<Data_type> var = new ArrayList<Data_type>(initial_capacity);
```

You cannot access ArrayList elements with direct indices: [...] But you can use many methods provided:

- `add(Type element)`, ← element must be of the same type
- `get(int index)`, ← get the element at the index
- `remove(int index)`,
- `indexOf(Type element)`,
- `set(int index, Type element)`,
- `size()`
Smiley[] smilies = new Smiley[3];
for (int i = 0; i < smilies.length; i++)
{
    smilies[i] = new Smiley();
}
smilies[0].color
    = Color.GREEN;
...

Arrays of objects

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>true</th>
<th>false</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREEN</td>
<td>BLUE</td>
<td>CYAN</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Declaring and creating 2D arrays

```java
int[][] table = new int[4][3];

or

int[][] table;
table = new int[4][3];
```
What is Package?

• A collection of classes grouped together **into a folder**
  – group related classes into one unit
  • We want to divide a big project into multiple components (at many levels)
  • The highest level is a package
  • Package -> Class -> Method

• See how Eclipse is divided into packages. Each package contains several tens of classes
What is Package?

- Package can be used to resolve the name clashes of classes

  - So many classes in a big project. Same name can be used in different context (or used by different programmers)
  - We have a self-defined Point class in Lab 6
  - There is another Point class in java.awt package
  - By putting them into different packages, we can control which one to use and how it is used
Throw and Catch

• In Java, the mechanism is called “Exception Handling”
  – **Throw an exception**: report a problem and asks for some code to handle it properly
  – **Catch an exception**: a piece of code dedicated to handle one or more specific types of problem
Declaring, Throwing, and Catching Exceptions

```java
method1() {
    try {
        invoke method2;
    } catch (Exception ex) {
        Process exception;
    }
}

method2() throws Exception {
    if (an error occurs) {
        throw new Exception();
    }
}
```

Overview of Exception handling
try {
    statements;
} catch(TheException ex) {
    handling ex;
} finally {
    finalStatements;
}
Text Files vs Binary Files

- Text file: a sequence of characters
- Binary file: pack values into binary representation

**FIGURE 10.2A** Text File and a Binary File Containing the Same Values

<table>
<thead>
<tr>
<th>A text file</th>
<th>A binary file</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 - 4 0 2 7 8 ...</td>
<td>12345 -4072 8 ...</td>
</tr>
</tbody>
</table>
Before Any File Operation

```java
import java.nio.charset.Charset;

... 
Charset charset = Charset.forName("US-ASCII");

Or
Charset charset = Charset.defaultCharset();
```

This creates an instance of charset to represent the character set in reading/writing files. (This tells Java which “language” we are speaking)
public static List<String> readAllLines(Path path, Charset cs) throws IOException
    
    Read all lines from a file.

Parameters:
    
    path - the path to the file
    cs - the charset to use for decoding

Returns:
    
    the lines from the file as a List;

Throws:
    
    IOException - if an I/O error occurs reading from the file or a malformed or unmappable byte sequence is read
    
    SecurityException - In the case of the default provider, and a security manager is installed, the checkRead method is invoked to check read access to the file.
Reading a File: Line by Line

```java
Charset charset = Charset.defaultCharset();
Path path = Paths.get("C:/data/students_name_list.txt");

try (BufferedReader reader = Files.newBufferedReader(path, charset)) {
    String line = null;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
        // do sth meaningful about this line
    }
} catch (IOException x) {
    System.err.format("IOException: %s%n", x);
}
```
Instead of doing:

```java
Files.newBufferedWriter(path, charset);
```

We can do:

```java
Files.newBufferedWriter(path, charset, openOption);
```

- Where `openOption` is a static member of `StandardOpenOption` class
The *is-a* relationship

- This inheritance relationship is known as an *is-a relationship*

- A Doctoral student *is a* Grad student
- A Grad student *is a* Student
- A Student *is a* Person

- Is a Person a Student?
  - Not necessarily!
Our general class is called a **base class**
- Also called a **parent class** or a **superclass**

Examples:
- Person, Transportation
A specialized class that inherits properties from a base class is called a **derived class**
- Also called a **child class** or a **subclass**

**Examples:**
- Student *is-a* Person
- Employee *is-a* Person
- Car *is-a* form of Transportation
- Animal *is-a* form of Transportation
Why is inheritance useful?

- Enables you to define shared properties and actions *once*
- Derived classes can perform the same actions as base classes without having to redefine the actions
  - If desired, the actions *can* be redefined
public class Derived_Class_Name extends Base_Class_Name
{
    Declaration_of_Added_Instance_Variables
    Definitions_of_Added_And_Overridden_Methods
}

public class Student extends Person
{
    // stuff goes here
}

• A derived (child) class inherits the public instance variables and public methods of its base (parent) class
private vs. public

• **private** instance variables and **private** methods in the base class are NOT inherited by derived classes

• This would not work:

```java
public Student(String stdName, int idNumber) {
    name = stdName; // ERROR! name is private to Person
    setID(idNumber);
}
```
Overriding vs. overloading

• If a derived class defines a method of the same name, same number and types of parameters, and same return type as a base class method, this is **overriding**

• You can still have another method of the *same name* in the same class, as long as its number or types of parameters are different: **overloading**
Recursion

• Whenever an algorithm has one subtask that is a smaller version of the entire algorithm’s task, it is said to be recursive.

• **Recursion**: you write a method to solve a big task, and the method invokes itself to solve a smaller subtask

• E.g., I want to eat 5 apples now. My subtask can be eating 4 apples, eating 3 apples, eating 2 apples, et.....

• To eat 5 apples, I can do:
  – Eat 3 apples + Eat 2 apples
  – Eat 1 apple + Eat 4 apples
Sequential Search

• Basic idea (from wiki)
  – For each item in the list:
    • if that item has the desired value, stop the search and return the item's location.
  – Return *Not Found*.

• Can you do better than this (by making it faster)?

• The general answer is no
  – No assumptions made on array
  – In worst case, have to examine each array element at least once
int binary_search(int A[], int key, int imin, int imax) {
    // test if search range is empty
    if (imax < imin) {
        return KEY_NOT_FOUND; // set is empty
    } else {
        // calculate midpoint to cut set in half
        int imid = midpoint(imin, imax);
        // three-way comparison
        if (A[imid] > key) // key is in lower subset
            return binary_search(A, key, imin, imid - 1);
        else if (A[imid] < key) // key is in upper subset
            return binary_search(A, key, imid + 1, imax);
        else // key has been found
            return imid;
    }
}
• Introduction to Sorting
  – Bubble sort
  – Selection sort
  – Merge sort

  – You should understand the idea behind bubble sort & selection sort
  – You should be able to understand the code given in slides (and know how to use the code in similar problems by making slight modification).