CMSC 132: OBJECT-ORIENTED PROGRAMMING II

Object-Oriented Programming Intro

Department of Computer Science
University of Maryland, College Park
Object-Oriented Programming (OOP)

- Approach to improving software
  - View software as a collection of objects (entities)

- Motivated by software engineering concerns
  - To be discussed later in the semester

- OOP takes advantage of two techniques
  - Abstraction
  - Encapsulation
Techniques – Abstraction

• Abstraction
  • Provide high-level model of activity or data

• Procedural abstraction
  • Specify what actions should be performed
  • Hide algorithms

• Data abstraction
  • Specify data objects for problem
  • Hide representation

• Abstract Data Type
  • Implementation independent interfaces
  • Data and operations on data
Techniques – Encapsulation

• Encapsulation
  • **Definition:** Hiding implementation details while providing an interface (methods) for data access
  • Allow us to use code without having to know its implementation
  • **Simplifies the process of code modification and debugging**
Abstraction & Encapsulation Example

• Abstraction of a Roster
  • Data
    • List of student names
  • Actions
    • Create roster
    • Add student
    • Remove student
    • Print roster

• Encapsulation
  • Only these actions can access names in roster

<table>
<thead>
<tr>
<th>ROSTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of names</td>
</tr>
<tr>
<td>create( )</td>
</tr>
<tr>
<td>addStudent( )</td>
</tr>
<tr>
<td>removeStudent( )</td>
</tr>
<tr>
<td>print( )</td>
</tr>
</tbody>
</table>
Java Programming Language

- Language constructs designed to support OOP
  - Interfaces
    - Specifies a contract
    - Provides abstract methods (no implementation)
  - Two views
    - Enforcing implementation of methods
    - Defining an IS-A relationship
  - Class
    - Implements/defines contract
    - Supports encapsulation of implementation (e.g., via private)
    - Class extending another class
      - Allows new class to inherit everything from original class
      - Defines an IS-A relationship
  - Class libraries designed using OOP principles
Object & Class

- **Class**
  - Blueprint for objects (of same type)
  - Exists at compile time

- **Object**
  - Abstracts away (data, algorithms) details
  - Encapsulates data
  - Instance exist at run time
Java Collections Framework

• Collection
  • Object that groups multiple elements into one unit
  • Also called container
  • **Example**: ArrayList

• Collection **framework** consists of
  • Interfaces
  • Implementations
Java Collections Framework

- **Collection** → Java Interface
  - See Java API entry for Collection
    - [http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)
  - **Example**: CollectionExample.java
- **Collections** → Class
  - [http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html](http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html)
About Style/Code

- Use Eclipse’s “Quick Fix”
- Use Eclipse’s source generation tools
  - Not for equals and hashCode methods
- Source ➔ “Organize Imports”
- Source ➔ Format
- About Eclipse Errors/Warnings
  - [http://www.cs.umd.edu/eclipse/other.html#errors-warnings](http://www.cs.umd.edu/eclipse/other.html#errors-warnings)
Iterator Interface

• Interface
  public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove();
  }

• Example usage
  ArrayList<String> L = new ArrayList<String>();
  L.add("Mary");
  L.add("Pete");
  Iterator<String> i = L.iterator();
  while (i.hasNext())
    System.out.println(i.next());
Enhanced For Loop

- Works for arrays and any class that implements the `Iterable` interface, including all collections
  - [http://docs.oracle.com/javase/6/docs/api/java/lang/Iterable.html](http://docs.oracle.com/javase/6/docs/api/java/lang/Iterable.html)
  - Has method `iterator()` returns `Iterator<T>` object
- For loop handles `Iterator` automatically
  - Test `hasNext()`, then invoke `next()`
- /* Iterating over a String array */
  ```java
  String[ ] roster = {"John", "Mary", "Alice", "Mark"};
  for (String student : roster)
    System.out.println(student);
  ```
Enhanced For Loop

```java
ArrayList<String> roster = new ArrayList<String>( );
roster.add("John");
roster.add("Mary");
/* Using an iterator */
for (Iterator<String> it = roster.iterator( ); it.hasNext( ); )
    System.out.println(it.next( ));
/* Using for loop */
for (String student : roster)
    System.out.println(student);
```
Generics (Motivating Example)

• Problem
  • Utility classes handle arguments as Objects
  • Objects must be cast back to actual class
  • Casting can only be checked at runtime

• Example
  class A { … }  
class B { … }  
List myL = new List();  
myL.add(new A());  // Add an object of type A

  …

  B b = (B) myL.get(0);  // throws runtime exception
                    // java.lang.ClassCastException
Solution (Generic Types)

- Generic types
  - Provides abstraction over types
  - Can parameterize classes, interfaces, methods
  - Parameters defined using \(<X>\) notation

- Examples
  - `public class foo<X, Y, Z> { … }`
  - `List<String> myNames = …`

- Improves
  - Readability & robustness

- Used in Java Collections Framework
Generics (Usage)

- Using generic types
  - Specify `<type parameter>` for utility class
  - Automatically performs casts
  - Can check class at compile time

- Example
  ```java
class A { ... }
class B { ... }
List<A> myL = new List<A>();
myL.add(new A()); // Add an object of type A
A a = myL.get(0); // myL element ⇒ class A
...
B b = (B) myL.get(0); // causes compile time error
```
**Autoboxing & Unboxing**

- Automatically convert primitive data types
  - Data value \(\Leftrightarrow\) Object (of matching class)
  - Data types & classes converted
    - Boolean, Byte, Double, Short, Integer, Long, Float

- Example
  ```java
  ArrayList<Integer> myL = new ArrayList<Integer>();
  myL.add(1);  // previously myL.add(new Integer(1));
  int y = mL.getFirst();
  //previously int y = mL.getFirst().intValue();
  ```

- **Example:** SortValues.java