CMSC 132: OBJECT-ORIENTED PROGRAMMING II

Java Language Constructs I

Department of Computer Science
University of Maryland, College Park

About Style/Code

- Let's use Eclipse's
  - Edit→Select All
  - Source→Format Element
- No need for { } for single statements
- Use Eclipse's "Quick Fix"
- Use Eclipse's source generation tools
  - Not for equals and hashCode methods

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
  - Has method iterator( ) returns Iterator<T> object
- For loop handles Iterator automatically
  - Test hasNext( ), then invoke next( )
- /* Iterating over a String array */

```
String[ ] roster = {"John", "Mary", ""Alice", "Mark"};
for (String student : roster)
    System.out.println(student);
```

Enhanced For Loop

```
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

```
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 ⇔ Object (of matching class)
  - Data types & classes converted
    - Boolean, Byte, Double, Short, Integer, Long, Float
- Example

```
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

Enumerated Types

- New type of variable with set of fixed values
  - Establishes all possible values by listing them
  - Supports values(), valueOf(), name(), compareTo()…
  - Can add fields and methods to enums
- **Example:** Color.java
- In Eclipse we define them as we defined classes
- When to use enums
  - Natural enumerated types – days of week, phases of the moon, seasons
  - Sets where you know all possible values
- **Example:** Deck.java
- **Example:** BoardCell.java