CMSC 132: Object-Oriented Programming II

Java Language Constructs

Department of Computer Science
University of Maryland, College Park
Review of Java Language Constructs

- Basic elements
  - Primitive types, variables, constants, operators
  - If-else, switch, while, for

- Classes
  - Object instances
    - Creating objects with new
  - Object references
    - The null reference
  - Instance data, class (static) data
  - Methods
    - Parameters, return values, polymorphism
Review of Java Language Constructs

Inheritance
- Base class, derived class, super
- Method overriding (vs. overloading)
- Abstract methods
- Up- and down-casting, getClass(), instanceof
  - avoid overuse of these... leads to bad designs

Interfaces

1D Arrays
- Creating, indexing

Exceptions
- Try-catch blocks
Java Language Constructs

- Iterator Interface
- Enhanced for loop
- Enumerated types
- Generics
- Autoboxing
- Comparable Interface
- Comparator Interface
**Iterator Interface**

**Interface**

```java
public interface Iterator {
    boolean hasNext( );
    Object next( );
    void remove( );    // optional, called once per next()
}
```

**Example usage**

```java
Iterator i = myCollection.iterator( );
while (i.hasNext( )) {
    myCollectionElem x = (myCollectionElem) i.next( );
}
```
Enhanced For Loop

- Works for arrays and any class that implements the `Iterable` interface, including all Collections
  - 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);
```
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

```java
public enum Color { Black, White } // new enumeration
Color myC = Color.Black;
for (Color c : Color.values()) System.out.println(c);
```

When to use enums

- Natural enumerated types – days of week, phases of the moon, seasons
- Sets where you know all possible values
Enumerated Types

From "Taming the Tiger" presentation by Joshua Bloch and Neal Gafter at Sun's 2004 Worldwide Java Developer Conference

```java
public class Card implements Serializable {
    public enum Rank { DEUCE, THREE, FOUR, FIVE, SIX,
                        SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE }
    public enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }
    private final Rank rank;
    private final Suit suit;
    private Card( Rank rank, Suit suit ) {
        this.rank = rank;
        this.suit = suit;
    }
    public Rank rank( ) { return rank; }
    public Suit suit( ) { return suit; }
    public String toString( ) { return rank + " of " + suit; }
}
```
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 ↔ 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)
Comparable Interface

- **Comparable**
  - `public int compareTo(Object o)`
  - `A.compareTo(B)` returns
    - Negative if `A < B`, 0 if `A == B`, positive if `A > B`

- **Properties**
  - Referred to as the class's *natural ordering*
  - Can sort using `Collections.sort()` & `Arrays.sort()`
    - Example: `Collections.sort(myList);`
  - Can use as keys in `SortedMap` & `SortedSet`
  - Consistency w/ `equals()` strongly recommended
    - `x.equals(y)` if and only if `x.compareTo(y) == 0`
  - Example (comparableExample)
Comparator Interface

Comparator

- public int compare(Object A, Object B)
  - Negative if A < B, 0 if A == B, positive if A > B

Properties

- Imposes total ordering on objects of a class
- Provide alternatives to natural ordering
- Supports generics
  - Example: class myC implements Comparator<Foo>{ … }
- Use as parameter for sort function
  - Example: Collections.sort(myFooList, new myC( ));

Example (comparatorExample)
2-D Arrays of Primitives

- Each row in two-dimensional array is an array
- Rows can have different lengths
- Defining a primitive array where rows have the same length

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

- Defining a primitive data array where rows have different lengths (ragged array)

```java
int[][] ragged = new int[2][];
ragged[0] = new int[3];
ragged[1] = new int[1];
```
2-D Arrays of Objects

- Each row in two-dimensional array is an array.
- Rows can have different lengths.
- Defining an array where rows have the same length:
  ```java
  String[][] data = new String[3][4];
  ```
- Important – Note we have created a 2-D array of references to String objects; no String objects yet exist.
- Can also create ragged arrays of objects.
- Example (Roster.java)