CSE 115/503

April 18– 22, 2011

Announcements

- Lab 8 started this week in recitation
- Lab 8 (continuation of Lab 7) will be due May 2nd for all sections
- Grades on UBLearns
  - If you have not come to see me about an issue, please do so.
- Exam 5 is Monday, April 25th
- Review for Exam 5 will be Wednesday, April 20th

Announcements - Lectures

- Friday, April 22 – no lecture

Collections

- If we want to be able to keep track of and use a group of objects, we need a way to store that group.
- In a single variable, we can only hold onto one object, so we need a variable that will hold onto an object that holds onto many other objects.

java.util.Collection

- Interface that is the root of the collection hierarchy.
- Interfaces tell us about functionality – in this case tell us what collections can do.
- There are other classes in java.util that implement this interface that we can create instances of.

Collection<E> - What is the <E>?

- The <E> represents a generic type
- Collections contain many objects, the <E> tells us what type of object is in the collection.
- When you are actually creating a collection, you would substitute the name of type for E.

Types

- What things qualify as a type?
  - Interface
  - Class
  - Abstract Class

Collections

- Declare a variable that holds onto a collection:
- java.util.Collection<E> name;

Creating a collection

- new java.util.LinkedList<E>();
- Remember that you can create an instance of any class that implements the Collection interface.
- Remember to assign that new instance to the variable you declared.

Putting stuff into a collection

- The add method allows us to insert elements into a collection.

Iterating over a collection

- If you want to "visit" every element in a collection, there is a piece of syntax built into Java to help us do that.
- It is called a for-each loop

For-each loop

- Syntax

```
for(TypeOfElementInCollection name: collection) {
   //what to do with each element
}
```

- TypeOfElementInCollection would match the E in the <E> when you declared and created the collection.

For-each loop

- name is a name you assign to the element. This is like a local variable. As the collection is iterated over, each element in the collection will be assigned to the variable name. This is what allows you to do things with each element.

For-each loop

- collection is the reference to the collection that you are iterating over. Often, it will be the name of the variable for the collection.
- Inside the loop's body, you would write the code that will be executed for every element in the collection.

Collection types

- LinkedLists, or more generally, Lists (List is an interface in java.util), represent one type of collections.
- These collections are often categorized as Bags.
- Bag collections are simply groups of elements. There may be an ordering of those elements, or there may not be.

Examples of Bags

- Real-world
  - Book bag
  - Piece of luggage
  - Grocery list
  - To-do list
  - Line at bank

Examples of Bags

- Computing
  - Linked List
  - Stack
  - Queue
  - Set
- We will not be covering these in depth. For purposes of collections we will need this semester, we will most often use a LinkedList when we need a Bag.

Collection Types

- The other general type of collection is called a Map
- Maps map a key to a value

Examples of Maps

- Real-world
  - Dictionary
  - Phone Book
  - Index in a book

Examples of Maps

- Computing
  - Hash map
  - Hash table
- We will be using a Hash Map this semester if we would like to map keys to values

Maps in Java

- There is an interface java.util.Map that is part of the collections framework, but Maps are not technically considered collections by Java.
- This means the methods on Maps are potentially different than those on Collections.
  - Meaning their names, parameters, and return types could be different

Declaring a HashMap

- java.util.HashMap<K, V> name;
- K is the type of the key, V is the type of the value.
- Any type can be a key or value, but keys need to ensure that they implement the methods hashCode and equals. Any standard object in Java implements these two methods, but classes you write may not.

Creating a HashMap

- new java.util.HashMap<K, V>();
- Remember to use this on the right hand side of an assignment if you want to maintain a reference to the hash map.

Inserting into a HashMap

- map.put(X, Y);
- Where map is the name of the variable referring to the map, and X and Y are the objects (object references) of what is to be inserted as the (key,value) pair.

Getting things out

- You would use the key to retrieve elements from a HashMap.

```
map.get(X);
```

- Would return the value associated with the key represented by the object reference X.

Iterating over a Map

- It is not as common to actually iterate over the elements in a hash map due to the very nature of what a mapping is.
- However, you are able to do it. You can separately iterate over the keys of the map or the values.

For-each on a Map's Keys

```
for(KeyType k: mapName.keySet()) {
   //what to do for each key
}
```

For-each on a Map's Values

```
for(ValueType v: mapName.values()) {
   //what to do for each value
}
```

Equals

- In an example in class, we used a method named *equals* to determine if a color was equal to the color red.
- *equals* is a method that you can write for objects that will help determine equality specific to a particular type.
- The *equals* method returns a boolean value.

Primitive Types

- Other types that are built into Java.
- Since Java 5, primitive types are more easily wrapped by their "object" counterparts and virtually anything that could have been done with primitives can now be done by using those objects.

Can't use primitives

- If you are trying to make a collection, you need to use a class name as the name inside the generic type (the <>), so you can not use a primitive type there.

What good are classes?

- Take a look – for the primitive type wrapper classes, quite a few have methods defined that are useful when working with that particular type of data.

Numbers

- Integer numbers
  - Primitive types: byte, short, int, long
  - Classes: Byte, Short, Integer, Long
- Floating point numbers
  - Primitive types: float, double
  - Classes: Float, Double

Operations on Numbers

- Addition
- Subtraction
- Multiplication
- Division
- Modulus
- Math class defines other operations

Numbers

- The number

  345
- is considered by Java to be of type int, but because of autoboxing and autounboxing, the following is valid:
- Integer number = 345;

Characters

- A single letter, digit, or other character.
- Primitive type
  - char
- Class type
  - Character

Boolean

- A variable whose type is boolean (primitive type) or Boolean (class type) can only have the values true or false.
- Useful in logic
- A boolean value is also produced as a result of the relational operators

Loops Revisited

- We have seen the for-loop and the for-each loop.
- The for-loop can be classified as a definite loop because it can usually be easily determined how many times it will execute.

Loops Revisited

- What if you don't know exactly how many times you want to execute, but rather you want to loop until some event happens.
- That is where an indefinite loop comes in. Java has a while-loop for this purpose.

While loops

- Syntax:

```
while (booleanExpression) {
    //code to be repeated
}
```

- The while loop will continue to execute as long as the booleanExpression is true.

What we need to program:

- Boehm-Jacopini Theorem
  - Sequencing
    - Ability to specify the order things will be executed in.
  - Selection
    - Choice
  - Repetition
    - Looping

Sequencing

- Involves the order lines of code will get executed in, including method calls and returning from those method calls.

Selection

- Java supports three different types of selection:
  - if and if-else statements
  - switch/case statements (not part of this course)
  - polymorphism (selection based on type)

Repetition

- Java supports five different types of repetition
  - For-loops
  - For-each loops
  - While-loops
  - Do-while loops (not covered in class)
  - Recursion (not covered in class)