CSE 115/503

March 22-26, 2010

Announcements

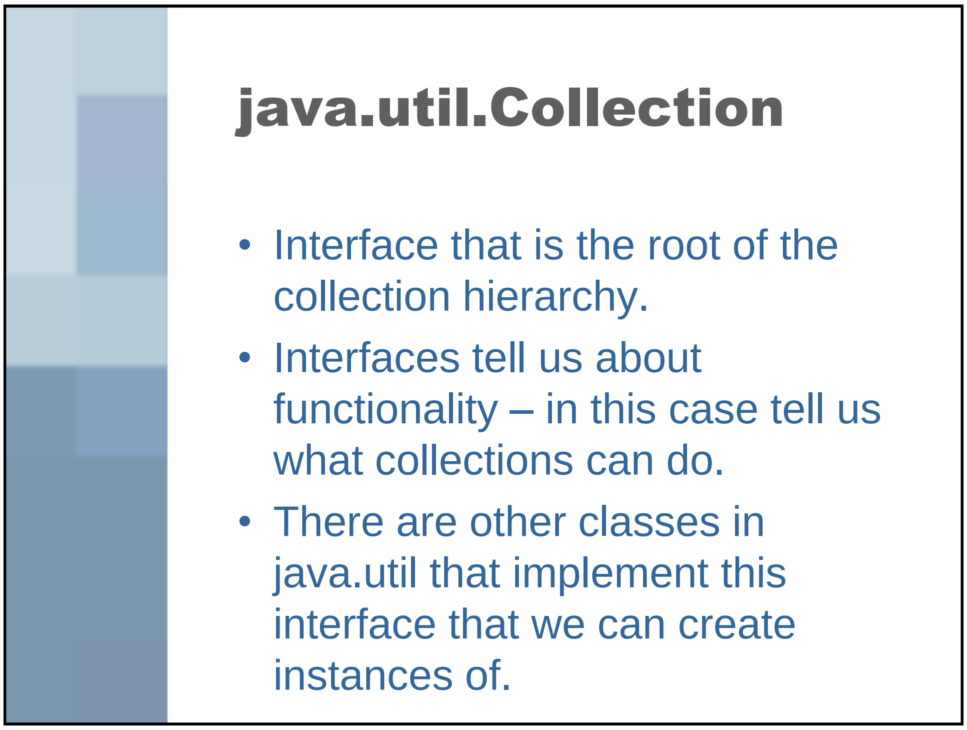
- Lab 6 posted this week – 1 week lab
- Exam 3 will be handed back in recitation this week unless
  - You are in A6
  - You did not indicate a recitation section
    - Adrienne has your exams in either of the above cases
- Exam 4 Review Monday 3/29
- Exam 4 Wednesday 3/31
  - There will be lecture after the exam
- No class on Friday 4/2
  - Adrienne will be out of town 4/1-4/4

Exam 3 Results

Min	6 (17 total F's)
Median	93.5
Average	82.69
Max	100 (80 total A's)

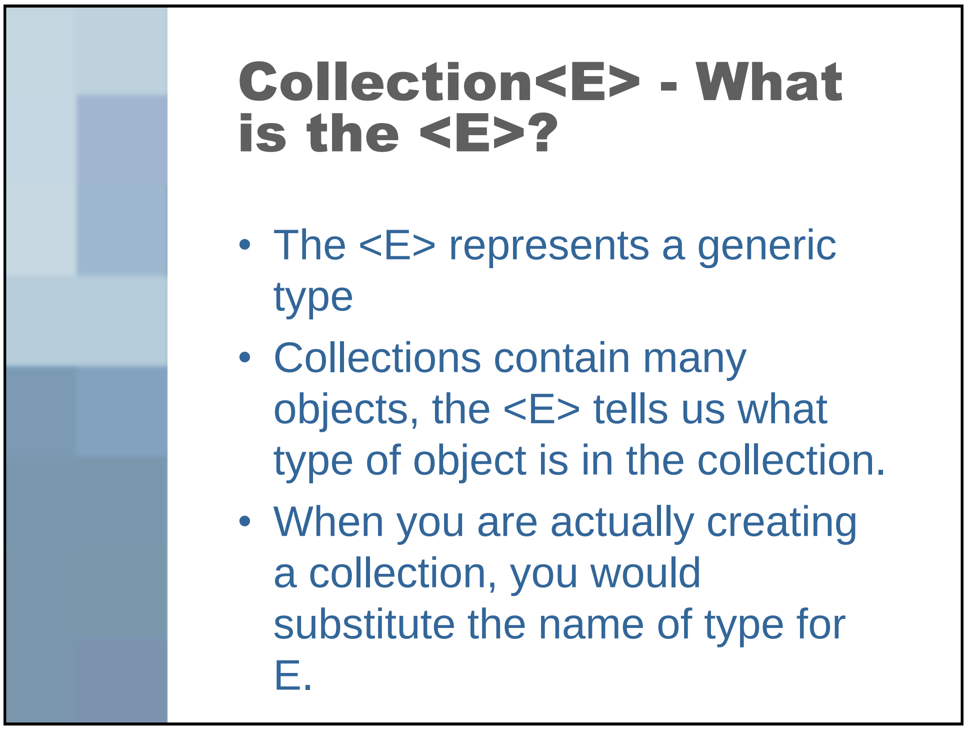
Collections

- 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

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

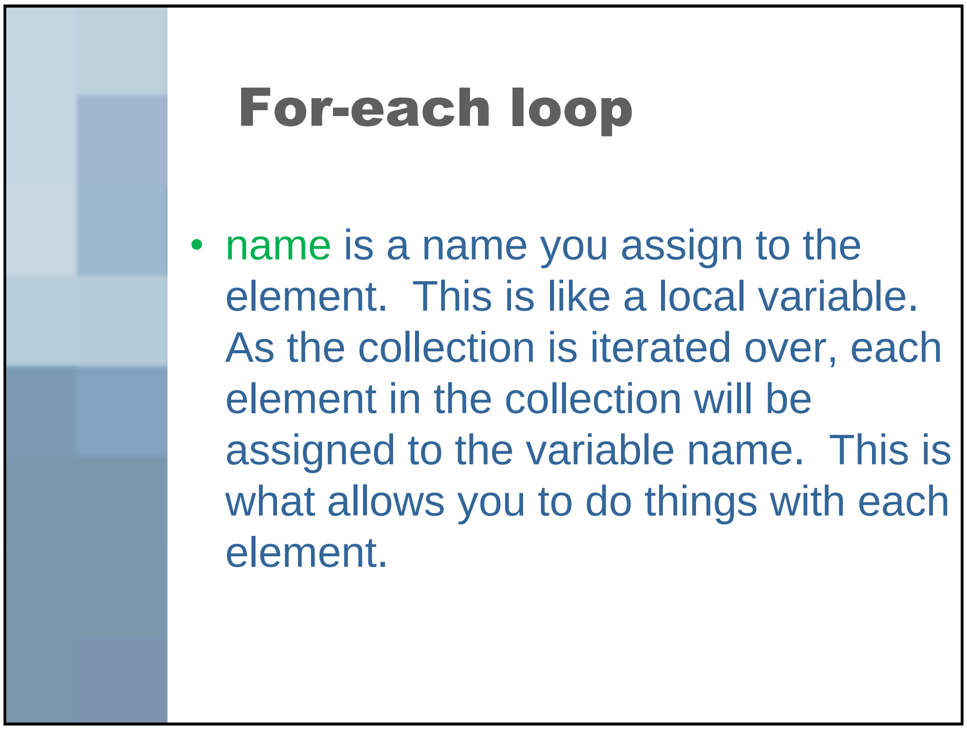
- If you want to “visit” every element in a collection, there is a piece of syntax built into Java to help us do that.
- It is called a for-each loop

For-each loop

- Syntax

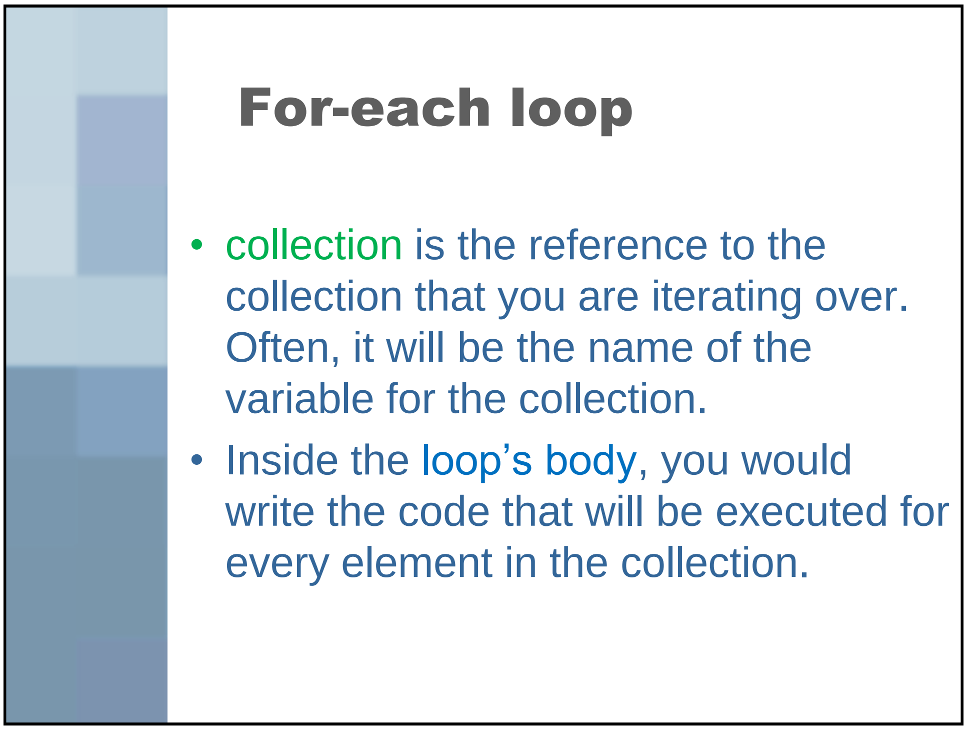
```
for(TypeOfElementInCollection name: collection) {
    //what to do with each element
}
```

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



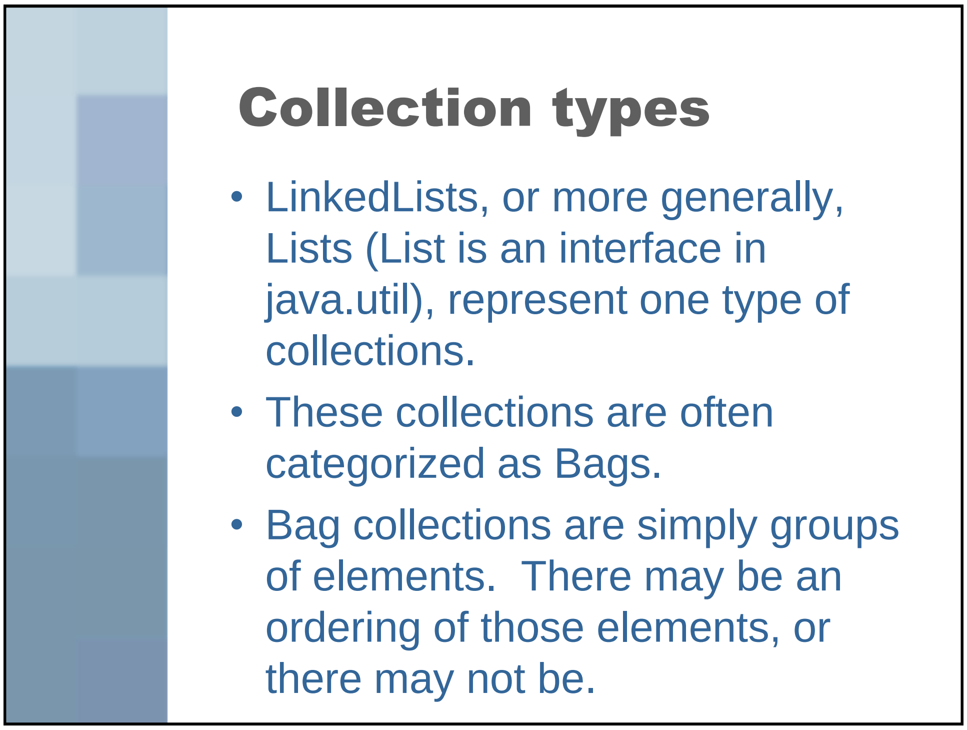
For-each loop

- 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.



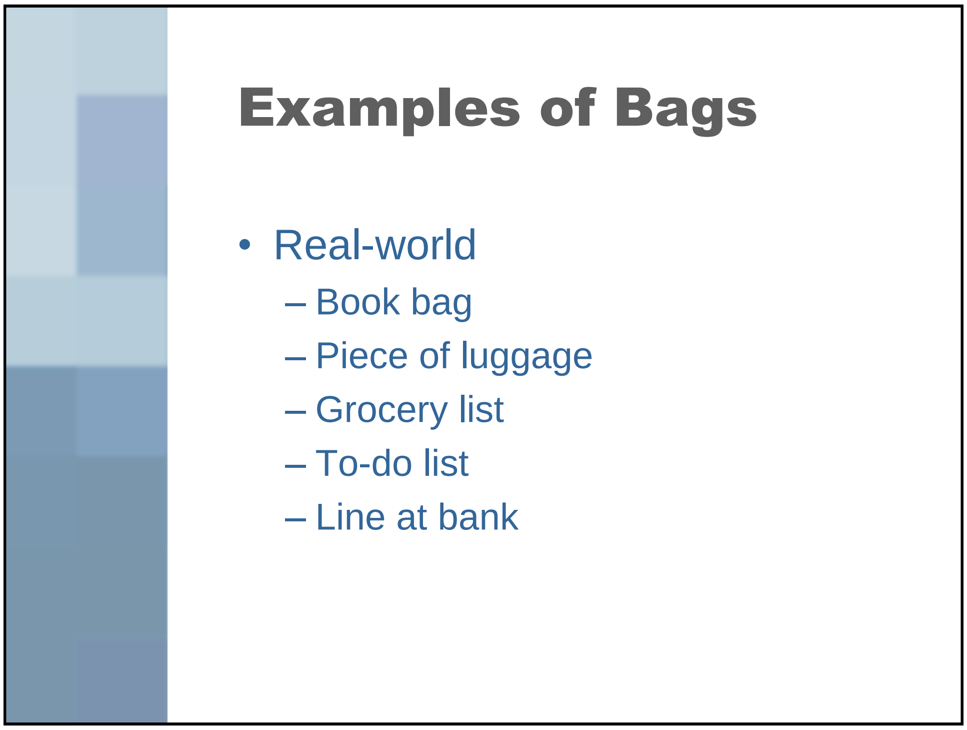
For-each loop

- 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

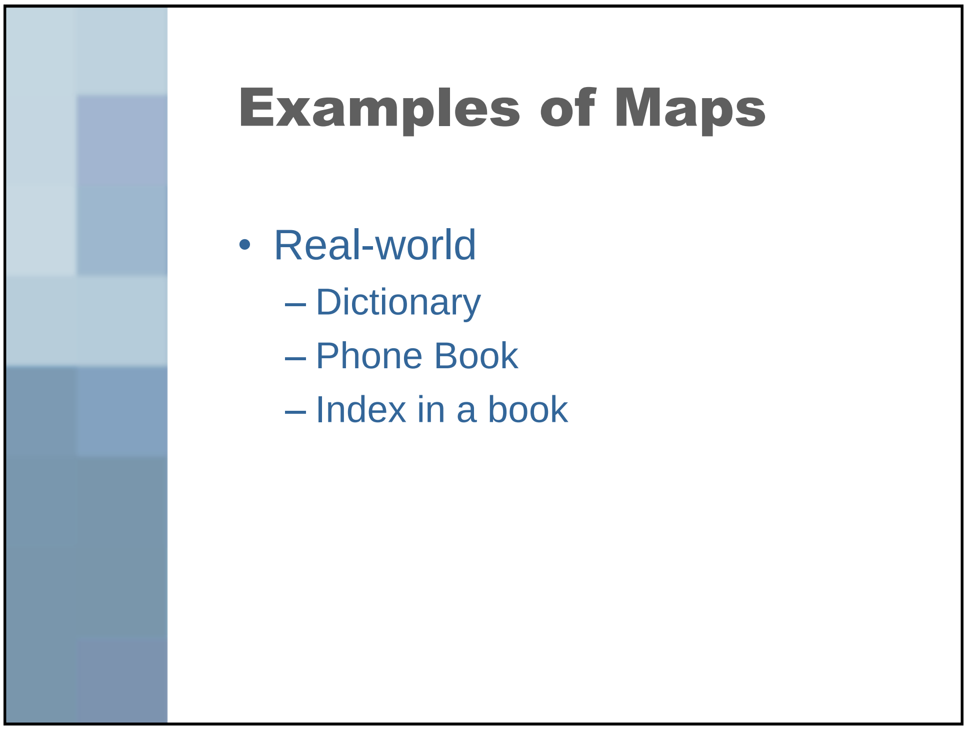
- Real-world
  - Book bag
  - Piece of luggage
  - Grocery list
  - To-do list
  - Line at bank

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.

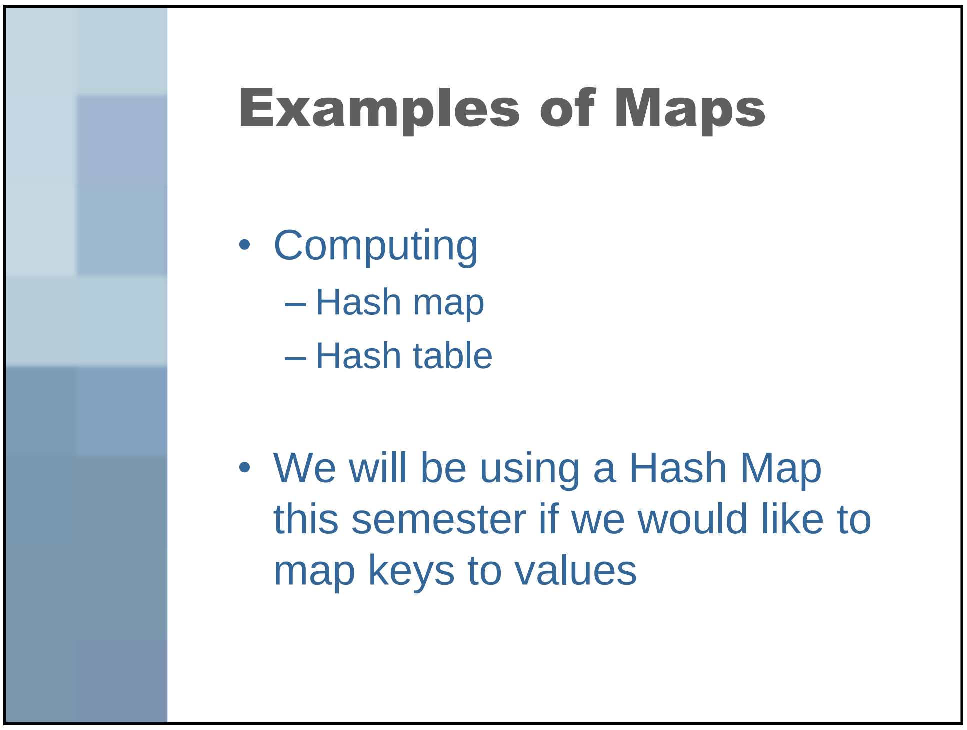
Collection Types

- 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

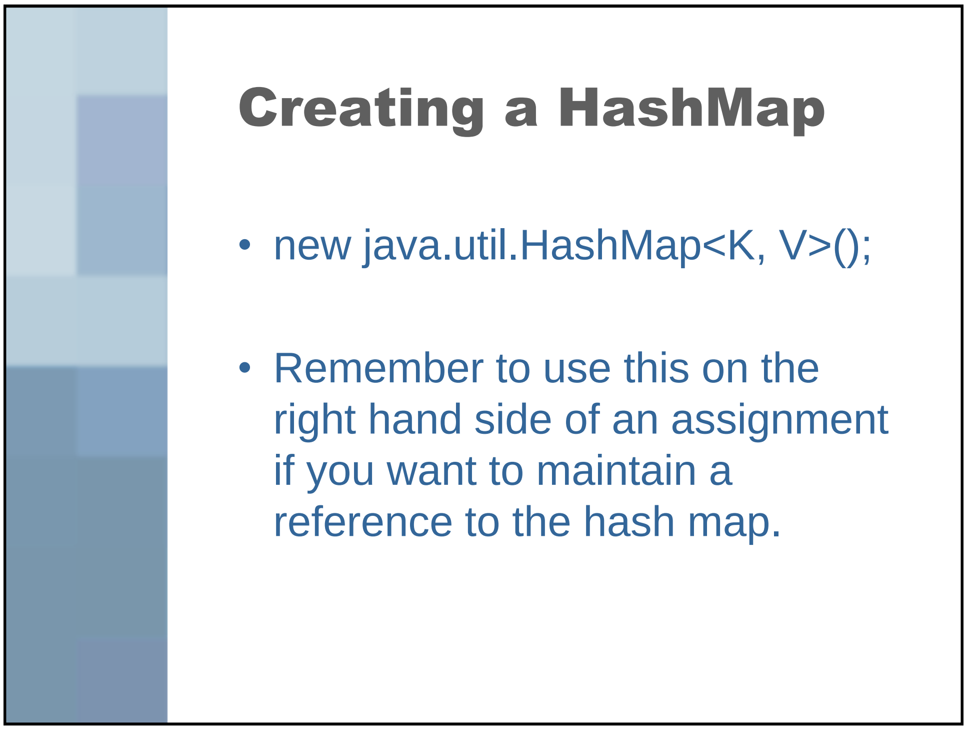
- Computing
  - Hash map
  - Hash table
- We will be using a Hash Map this semester if we would like to map keys to values

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

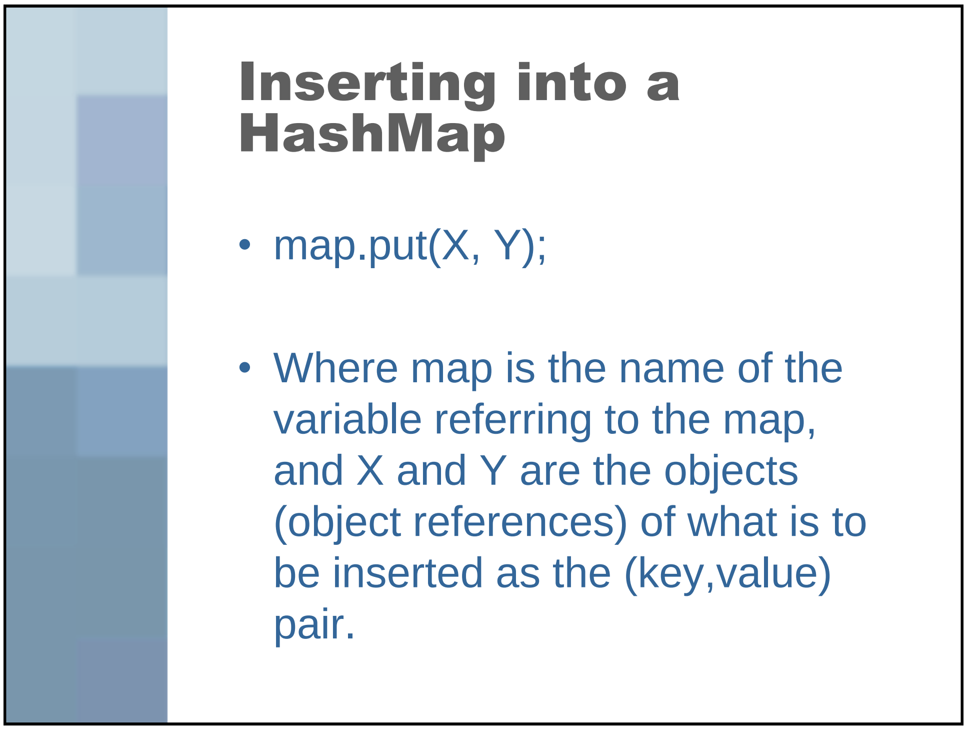
Declaring a HashMap

- 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
}
```

Repetition

- The ability for a program to repeat a task
- Java has five different repetition mechanisms built in
  - Three types of general purpose loops
    - for, while, do-while
  - One type of special purpose loop
    - for-each loop
  - Recursion

Repetition – for loop

- Syntax

```
for(initialization; expression; increment) {
   //code to be repeated
}
```

- initialization – typically we create a loop counter and assign it an initial value.

Repetition – for loop

- Syntax

```
for(initialization; expression; increment) {
   //code to be repeated
}
```

- expression – this expression is a boolean expression (one that evaluates to true or false) and usually represents the end condition for the counter variable.

Repetition – for loop

- Syntax

```
for(initialization; expression; increment) {
   //code to be repeated
}
```

- increment – change the value of the loop counter by some increment (or decrement)

Repetition – for loop

- Syntax

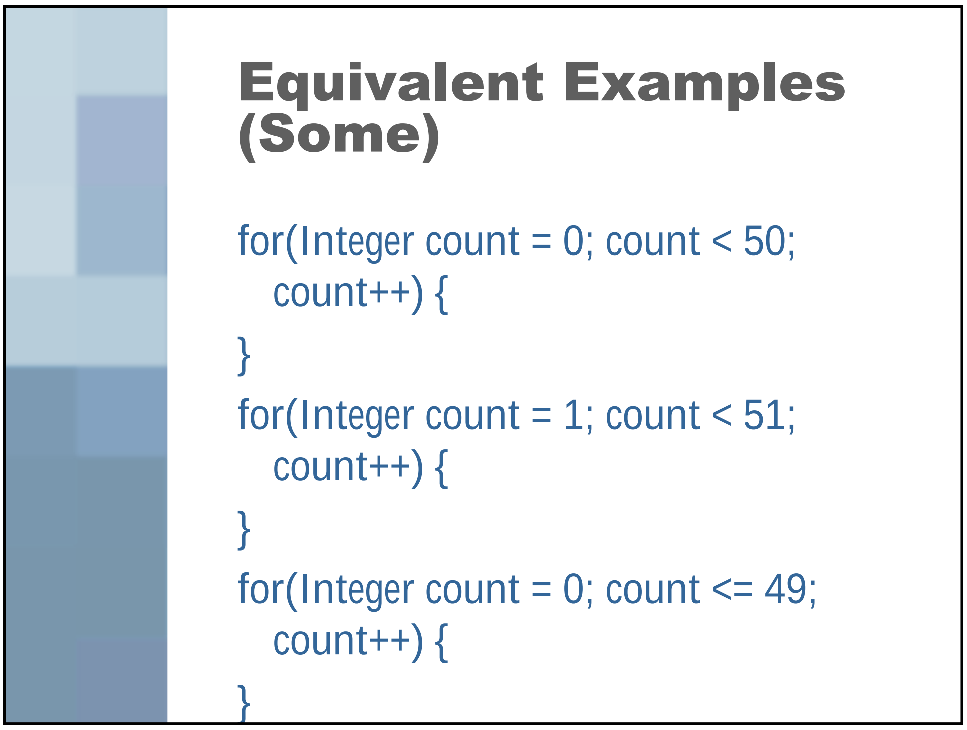
```
for(initialization; expression; increment) {
   //code to be repeated
}
```

- code to be repeated – the code we want repeated each time. This is called the body of the loop.

Repetition - Example

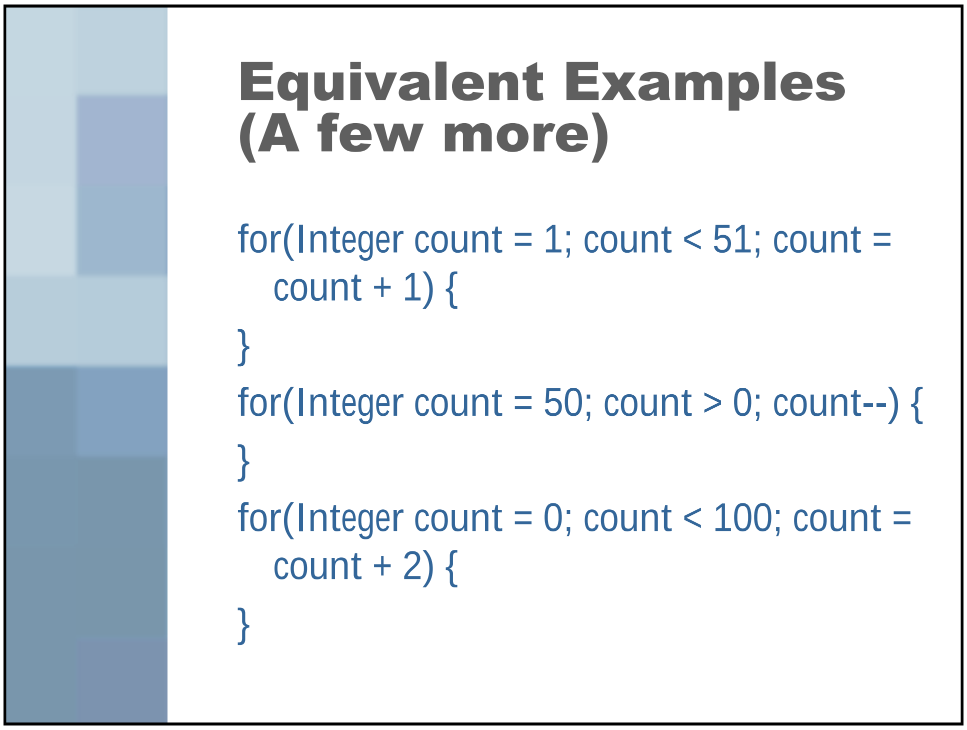
- Write a loop that executes 50 times

```
for(Integer count = 1; count <=50; count++) {
}
```



Equivalent Examples (Some)

```
for(Integer count = 0; count < 50;
   count++) {
}
for(Integer count = 1; count < 51;
   count++) {
}
for(Integer count = 0; count <= 49;
   count++) {
}
```

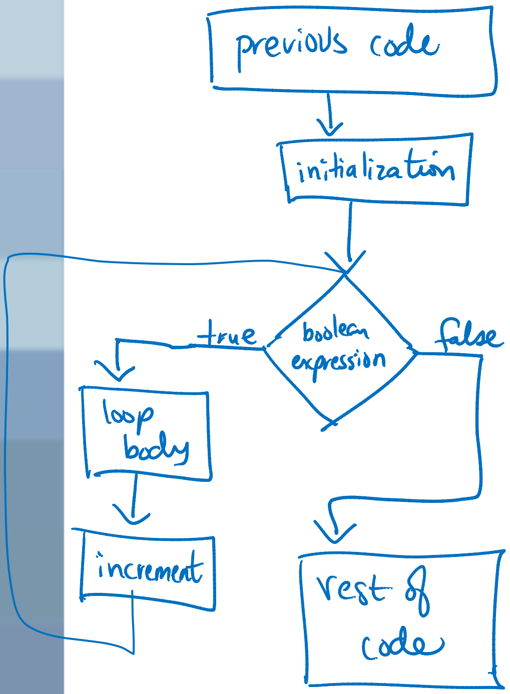


Equivalent Examples (A few more)

```
for(Integer count = 1; count < 51; count =
   count + 1) {
}
for(Integer count = 50; count > 0; count--) {
}
for(Integer count = 0; count < 100; count =
   count + 2) {
}
```

For-loop (Execution)

- The order the parts of the loop are executed in are perhaps best described in terms of a diagram.



Selection

- Selection is the ability to choose in a program.
- Java has three built-in mechanisms for selection
  - If-statements (including if/else statements)
  - Switch/case statements
  - Polymorphism

If-statement (Simple)

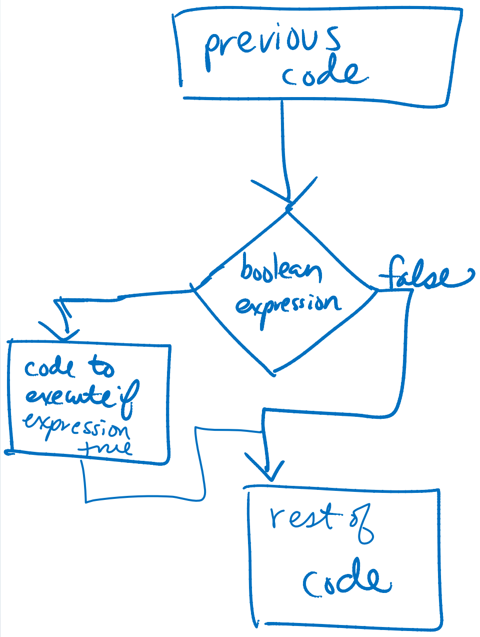
- Syntax

```
if(booleanExpression) {
   //code to be executed if booleanExpression is true
}
```

- A booleanExpression is an expression whose result is a boolean value (either true or false).

If-statement (Execution)

- Again, a diagram indicating the execution of an if-statement.



Lecture Code

- Please make sure to look at Lab4Modified, Lab4ModifiedVersion2, and Lab4ModifiedVersion3 to see various examples of these ideas

Equals

- In version 3, 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.