Java - Collections and Exceptions
Horstmann Chapters 1.8, 1.11 and 8.3

CS 4354
Summer II 2016
Jill Seaman

Collections in Java

• A collection is a data structure for holding elements
• java.util.Collection<T> is an interface implemented by many classes in Java. It has 3 extended interfaces:
  ✦ List<T> implemented by ArrayList<T> and LinkedList<T>, etc.
  ✦ Set<T> implemented by HashSet<T> and others
  ✦ Queue<T> implemented by PriorityQueue<T> and others

• Some methods in the Collection interface:
  ✦ isEmpty(), contains(e), add(e), remove(e), iterator()

Maps in Java

• A map is an object that associates keys with values.
• A map cannot contain duplicate keys; each key can map to at most one value.
• java.util.Map<K,V> is an interface implemented by many classes in Java
  ✦ HashMap<K,V>, Hashtable<K,V>
  ✦ TreeMap<K,V>
• Some methods in the Map interface:
  ✦ isEmpty(), containsKey(e), put(k,v), get(k), remove(k)
  ✦ values(): Collection<V>, keySet(): Set<K>

Diagram of Collections and Maps in Java
Linked Lists in the Java Library

- An linked list supports efficient insertion and removal at any location:

```
  ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼
  ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼
    New value
```

- java.util.LinkedList<T> is an class that implements List<T>
  ✦ void add(T e) appends to the end of the list
  ✦ T get(int i) and void set(int i, T e) are supported, but not efficient. Each call traverses the list.
  ✦ Use an iterator to access elements in the middle.

Iterators in Java

- An iterator is an object that cycles through all the elements in a collection. It points to an element of the collection.
  ✦ java.util.Iterator<T> is an interface with the following methods:
    ✦ public T next() returns the next element in the collection (and advances)
    ✦ public boolean hasNext() returns true if next() is not done.
    ✦ public void remove() (Optional) removes the last element returned by next.
  - You can get Iterators from Collections (and Maps):
    ✦ ArrayList<Double> x = new ArrayList<Double>
      Iterator<Double> it = x.iterator();
    ✦ HashMap<String,Double> hm = new HashMap<String,Double>
      Iterator<Double> it = hm.values().iterator();

Collections and Iterators: example

```java
public class ListIteratorTester {
    public static void main(String[] args) {
        LinkedList<String> countries = new LinkedList<String>();
        countries.add("Belgium");
        countries.add("Italy");
        countries.add("Thailand");
        Iterator<String> iterator = countries.iterator();
        while (iterator.hasNext()) {
            String country = iterator.next();
            System.out.println(country);
        }
        System.out.println();
        // Or use a for each loop
        for(String country : countries)
            System.out.println(country);
        System.out.println();
        // An Iterator can also remove elements:
        iterator = countries.iterator(); //reset to first element
        iterator.next();
        iterator.remove(); //removes second element
    }
}
```

Exceptions: Error Handling in Java

- Run time errors
  ✦ It is difficult to recover gracefully from run-time errors that occur in the middle of a program.
  ✦ At the point where the problem occurs, there often isn’t enough information in that context (the method) to resolve the problem.
  ✦ In Java, that method hands off the problem out to a higher context (a calling method) where someone is qualified to make the proper decision.

- If the error can be resolved in the immediate context where it occurs, it is NOT called an exception.
Exception semantics - 1

• When an error occurs inside a method, the method creates an exception object.
  ✦ could be in a library method or a user-defined method
• Reporting and exception it to the runtime system is called throwing an exception.
• When a method throws an exception,
  ✦ the current path of execution is interrupted, and
  ✦ the runtime system attempts to find an appropriate place to continue executing the program.

Exception semantics - 2

• The runtime system searches the call stack for an appropriate exception handler
  ✦ the call stack: the list of methods that have been called and are waiting for the current method to return.
  ✦ A calls B that calls C that calls D: The call stack contains A, B, C and D with D on the top.
• The runtime system is looking for a previous method call that is embedded in a block that has an exception handler associated with it.
  ✦ It starts at the top of the call stack and goes down (in reverse order in which the methods were called)

Exception semantics - 3

• The runtime system is searching for an appropriate exception handler
  ✦ An exception handler is considered appropriate if the type of the exception object thrown matches the type that can be handled by the handler
• The first exception handler encountered that matches the exception is said to catch the exception.
• If the runtime system exhaustively searches all the methods on the call stack without finding an appropriate exception handler, the runtime system terminates the program.
  ✦ And usually the exception is output to the screen

Exception syntax: how to throw an exception

• To throw an exception, use the keyword throw.
• To create an exception, use the appropriate constructor.
  if (t==null)
      throw new NullPointerException();
• Exception classes can be found in the API website: see java.lang.Exception
Exception syntax: how to catch an exception

- To catch an exception, use the try-catch block.
- Surround the code that might generate an exception in the try
- Make an exception handler (a catch clause) for every type of exception you want to catch.

```java
try {
    // Code that calls methods that might throw exceptions
    catch(Type1 id1) {
        // Handle exceptions of Type1
    }
    catch(Type2 id2) {
        // Handle exceptions of Type2
    }
    catch(Type3 id3) {
        // Handle exceptions of Type3
    }
    // etc...
}
```

Each catch clause is like a little method that takes one argument of a particular type.
- The parameters (id1, id2, and so on) can be used inside the handler, just like a method argument.
- If the handler catches an exception, its catch block is executed, and the flow of control proceeds to the next statement after (outside) the try/catch.
  - only the first matching catch clause is executed.

The exception specification: being civil

- In Java, you are (strongly!) encouraged to inform the client programmer, who calls your method, of the exceptions that might be thrown from your method.
  - Then the caller can know exactly what catch clauses to write to catch all potential exceptions.
- The exception specification states which exceptions are thrown by a method.
  - void f() throws TooBig, TooSmall, DivZero { //...}
  - Also use the @throws tag in the javadoc comment to describe these in more detail (when/why each one is thrown).
- Catch or specify requirement: If the method throws exceptions, it must handle them or specify them in the signature.
  - Otherwise it’s a compiler error.
Catch or Specify: example

```java
public class ListOfNumbers {
    private ArrayList<Integer> ints;
    private static final int SIZE = 10;
    public ListOfNumbers () {
        ints = new ArrayList<Integer>(SIZE);
        for (int i = 0; i < SIZE; i++) {
            ints.add(i);
        }
    }
    public void writeList()  {
        PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + ints.get(i));
        }
        out.close();
    }
}
```

ListOfNumbers.java:16: error: unreported exception IOException;
must be caught or declared to be thrown
(PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));)

Catch or Specify: solution 1

```java
public class ListOfNumbers {
    private ArrayList<Integer> ints;
    private static final int SIZE = 10;
    public ListOfNumbers () {
        ints = new ArrayList<Integer>(SIZE);
        for (int i = 0; i < SIZE; i++) {
            ints.add(i);
        }
    }
    public void writeList()  throws IOException  {
        PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + ints.get(i));
        }
        out.close();
    }
}
```

This compiles with no errors.

Catch or Specify: solution 2

```java
public void writeList()  {
    PrintWriter out = null;
    try {
        out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + ints.get(i));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    if (out != null)
        out.close();
}
```

This compiles with no errors.

Runtime Exceptions: an exception to the rule

- RuntimeExceptions are a special (sub)class of Exceptions.
  - They are thrown automatically by Java in certain contexts
  - This is part of the standard run-time checking that Java performs for you
- These exceptions are “unchecked exceptions”, they do not need to conform to the “Catch or specify rule.
  - Methods are not required to indicate if they might throw one
  - Methods are not required to try to catch them
- What if they are not caught?
  - If a RuntimeException gets all the way out to main() without being caught, printStackTrace() is called for that exception as the program exits
You can create your own exceptions

- If one of the Java Exceptions is not appropriate for your program, you can create your own Exception classes
  ✦ The class must inherit from an existing exception class, preferably one that is close in meaning to your new exception.

```java
class SimpleException extends Exception {}
class SimpleExceptionDemo {
    public void f() throws SimpleException {
        System.out.println("Throw SimpleException from f()");
        throw new SimpleException();
    }
}
public class DemoDriver {
    public static void main(String[] args) {
        SimpleExceptionDemo sed = new SimpleExceptionDemo();
        try {
            sed.f();
        } catch(SimpleException e) {
            System.err.println("Caught it!");
        }
    }
}
```