CS1004: Intro to CS in Java, Spring 2005
Lecture #25: Finishing Up Java
Janak J Parekh
janak@cs.columbia.edu

Administrivia
- HW#6 late day policy: we'll allow one late day

Today’s lecture
- Essentially the last lecture on Java; we'll touch upon it again a few times, but next week we'll focus on a few last interesting theory topics
- Today’s lecture will be a smorgasboard of topics from chapter 6 and 7
- Due to the lack of time, we’re removing GUI programming as required reading for chapters 5 through 7
  - Some other minor topics also removed; check syllabus
**ArrayList Efficiency**

- The **ArrayList** class is implemented using an underlying array.
- The array is manipulated so that *indexes remain continuous* as elements are added or removed.
- The **size()** method returns the number of actual objects in the ArrayList, and the code prevents you from accessing empty cells.
- If elements are added to and removed from the end of the list, this processing is fairly efficient.
- If elements are inserted and removed from the front or middle of the list, the remaining elements are shifted.

**The Iterator Interface**

- Recall that an iterator is an object that provides a means of processing a collection of objects one at a time.
- An iterator is created formally by implementing the **Iterator** interface, which contains three methods: **hasNext**, **next**, and **remove**.
- By having a class implement the **Iterator** interface, you can use the “compact” version of the for loop.
- **ArrayList** implements **Iterator**, so you can use the compact for loop.

**Enumerated Types**

- Earlier, we introduced *enumerated types*, which define a new data type and list all possible values of that type.
- Enums actually define a special class with those values as constants.
- You can set up special constructors and methods.
- We could have used enums for Rock-Paper-Scissors.
- You can use enums for Suits – optional.
Enumerated Types

- Every enumerated type contains a static method called `values` that returns a list of all possible values for that type.
- The list returned from `values` is an iterator, so a `for` loop can be used to process them easily.
- A carefully designed enumerated type provides a versatile and type-safe mechanism for managing data.

Parameter Passing

- Another important issue related to method design involves parameter passing.
- Parameters in a Java method are `passed by value`.
- A copy of the actual parameter (the value passed in) is stored into the formal parameter (in the method header).
- Therefore passing parameters is similar to an assignment statement.
- A quick example…

Passing Objects to Methods

- When an object is passed to a method, the actual parameter and the formal parameter become aliases of each other, because a copy of the `reference` is made.
- What a method does with a parameter may or may not have a permanent effect (outside the method).
- Note the difference between changing the internal state of an object versus changing which object a reference points to.
Method Overloading

- Method overloading is the process of giving a single method name multiple definitions
- If a method is overloaded, the method name is not sufficient to determine which method is being called
- The signature of each overloaded method must be unique
- The signature includes the number, type, and order of the parameters

```java
float tryMe(int x)
    return x + .375;
}

float tryMe(int x, float y)
    return x * y;
}
```

Method Overloading

- The compiler determines which method is being invoked by analyzing the parameters

```java
float tryMe(int x)
    return x + .375;
}

float tryMe(int x, float y)
    return x * y;
}
```

Method Overloading

- The `println` method is overloaded:
  - `println (String s)`
  - `println (int i)`
  - `println (double d)`
  - and so on...
- The following lines invoke different versions of the `println` method:
  ```java
  System.out.println("The total is:");
  System.out.println(total);
  ```
Overloading Methods

- The return type of the method is not part of the signature
- That is, overloaded methods cannot differ only by their return type
- Constructors can also be overloaded
- Overloaded constructors provide multiple ways to initialize a new object

Testing

- Testing can mean many different things
- At minimum, run a completed program with various inputs
- It also includes any evaluation performed by human or computer to assess quality
- Some evaluations should occur before coding even begins
- The earlier we find an problem, the easier and cheaper it is to fix

Testing

- The goal of testing is to find errors
- We can never really be sure that all errors have been eliminated
- So when do we stop testing?
  - Conceptual answer: Never
  - Snide answer: When we run out of time
  - Better answer: When we are willing to risk that an undiscovered error still exists
Reviews

- A *review* is a meeting in which several people examine a design document or section of code.
- It is a common and effective form of human-based testing.
- Presenting a design or code to others:
  - makes us think more carefully about it
  - provides an outside perspective
- Reviews are sometimes called *inspections* or *walkthroughs*.
- Not for this class, though.

---

Test Cases

- A *test case* is a set of input and user actions, coupled with the expected results.
- Often test cases are organized formally into *test suites* which are stored and reused as needed.
- For medium and large systems, testing must be a carefully managed process.
- Many organizations have a separate Quality Assurance (QA) department to lead testing efforts.

---

Defect and Regression Testing

- *Defect testing* is the execution of test cases to uncover errors.
- The act of fixing an error may introduce new errors.
- After fixing a set of errors we should perform *regression testing*—running previous test suites to ensure new errors haven’t been introduced.
- It is not possible to create test cases for all possible input and user actions.
- Therefore we should design tests to maximize their ability to find problems.
Black-Box Testing

- In black-box testing, test cases are developed without considering the internal logic.
- They are based on the input and expected output.
- Input can be organized into equivalence categories.
- Two input values in the same equivalence category would produce similar results.
- Therefore, a good test suite will cover all equivalence categories and focus on the boundaries between categories.

White-Box Testing

- White-box testing focuses on the internal structure of the code.
- The goal is to ensure that every path through the code is tested.
- Paths through the code are governed by any conditional or looping statements in a program.
- A good testing effort will include both black-box and white-box tests.

Next time

- Finish theory topics for the semester.