Intro to Computer Science II

CS112-2012S-05

I/O and ArrayList

David Galles

Department of Computer Science
University of San Francisco

05-0: More on I/O

- Lots of ways to use Scanner class
  - Always get more infomation by Googling 'Scanner Java'
  - Works for just about any Java library class – first hit will be the Oracle page
- Read from a file in addtion to reading from the keyboard

```
Scanner sc = new Scanner(new File("myNumbers"));
```

- method nextLine() can get the next line of input in one string
- method hasNextLine() can determine if there is a next line to read

05-1: More on I/O

```
Scanner scan = new Scanner(new File("myNumbers"));
while (scan.hasNextLine())
{
  String nextLine= scan.nextLine();
  // Do something with the next line of input
}
```

05-2: More on I/O

- Creating a new File may cause an error
  - File might not exist
- Constructor is tagged as a method that may cause an error
- We need to either handle the error (we'll do this later), or flag our method as one that can cause an error
- This kind of an error is called an exception
  - Example with Eclipse

05-3: More on I/O

```
import java.io.File;
import java.util.Scanner;
import java.io.IOException;


public static void main(String args[]) throws IOException
{
  Scanner scan = new Scanner(new File("myNumbers"));
  while (scan.hasNextLine())
  {
     String nextLine= scan.nextLine();
     // Do something with the next line of input
  }
}
```

05-4: More on I/O

- Class PrintStream
  - Used to print to a file
  - Works just like System.out
  - Constructor takes a filenmame (technically a filePath)

05-5: More on I/O

```
import java.io.File;
import java.util.Scanner;
import java.io.IOException;


public static void main(String args[]) throws IOException
{
  Scanner S = new Scanner(new File("test"));
  PrintStream ps = new PrintStream("testout");
  while (S.hasNextLine())
  {
    String next = S.nextLine();
    String upper = next.toUpperCase();
    ps.println(upper);
  }
  ps.close();
  S.close();
}
```

05-6: More on Strings

- Lab3
  - Replace strings within a file
  - Can use replaceAll method of string
    - Could replace substrings you don’t want to replace: cat in scat
    - replaceAll is acceptable, even with this bug
    - For a slightly more robust version, use split method of string (need to do some work on your own)

05-7: ArrayLists

- Arraylists are similar to python lists
  - Class, with data and methods
  - Many methods are similar to python list methods
  - Unlike python, need to declare type of elements in the list
  - Can only store class types (no built-in types – no int, bool, etc)

05-8: ArrayLists

- ArrayList declaration syntax is a little odd
- ArrayList<TypeName>

```
ArrayList<String> myList = new ArrayList<String>()
```

05-9: ArrayLists

- ArrayList Methods:
  - add(E element): Add element to end of list
  - add(E element, int index): Add element at location ’index’ (shifting elements to the right)
  - get(int index): Get element at location ’index’
  - set(int index, E element): Set element at location ’index’ to be ’element’
  - remove(int index): Remove the element at location ’index’ (shifting elements to left)
  - size(): Return number of elements in the list
  - Fore more, google “ArrayList java” to get oracle docs (example in class)

05-10: ArrayLists

```
ArrayList<String> myList = new ArrayList<String>();
myList.add("Cat");
myList.add("Dog");
myList.add("Mouse");
for (int i = 0; i < myList.size(); i++)
{
    System.out.println(myList.get(i));
}
```

05-11: ArrayLists

- Write a method that takes as an input parameter an unsorted ArrayList of strings and returns the largest (last in lexigraphic order) string

```
String findLargest(ArrayList<String> list)
{
   // Fill me in!
}
```

05-12: ArrayLists

- Write a method that takes as an input parameter an unsorted ArrayList of strings and returns the largest (last in lexigraphic order) string

```
String findLargest(ArrayList<String> list)
{
   if (list.size() == 0)
   {
      return "";
   }
   String largest = list.get(0);
   for (int i = 1; i < list.size(); i++)
   {
      if (largest.compareTo(list.get(i)) < 0)
      {
         largest = list.get(i);
      }
   }
   return largest;
```

05-13: ArrayLists

- Write a method that takes as an input parameter a sorted ArrayList of strings and a string to insert, and inserts the new string into the ArrayList in order

```
void insertIntoList(ArrayList<String> list, String elem)
{
   // Fill me in!
}
```

05-14: ArrayLists

- Write a method that takes as an input parameter a sorted ArrayList of strings and a string to insert, and inserts the new string into the ArrayList in order

```
void insertIntoList(ArrayList<String> list, String elem)
{
   int index = 0;
   while(index < list.size() && list.get(index).compareTo(elem) < 0)
   {
      index++;
   }
   list.add(index, elem);
}
```

05-15: ArrayLists

- In the above method, what happens to duplicates?
- Next lab, handle duplicates in a different way

05-16: Wrapper Classes

- There are a number of “Collection classes, ” Like the ArrayList class, that can only store other classes as data members
- What if we wanted to store ints, or floats in one of these collections?
  - What could we do?

05-17: Wrapper Classes

- Create a class that just stores a single int
- Integer class
- Constructor takes a single int
- intValue() method used to retrieve value
- immutable (like Strings!)

05-18: Integer

```
ArrayList<Integer> intList = new ArrayList<Integer>();
for (int i = 0; i < 10; i++)
{
   intList.add(new Integer(i*3));
}
for (int i = 0; i < 10; i++)
{
   System.out.println(intList.get(i).intValue());
}
```

05-19: Autoboxing

- The following could seems like it shouldn't work
- intList.add takes an Integer as an input parameter, not an int

```
ArrayList<Integer> intList = new ArrayList<Integer>();
for (int i = 0; i < 10; i++)
{
   intList.add(i*3);
}
for (int i = 0; i < 10; i++)
{
   System.out.println(intList.get(i));
}
```

05-20: Autoboxing

- To make life easier, if a method takes a parameter an Integer, and you pass it an int, then it is automatically “boxed” into an Integer
  - The code “new Integer(...)” is implicitly added
- If a method takes as a parameter an int, and you pass it an Integer, then it is automatically “unboxed” into an int
  - The code “.intValue()” is addedx

05-21: Autoboxing

```
Integer myint = new Integer(5);
int builtinInt = myint + 4;
myint = builtinInt + 5;
```

- Equivalent to:

```
Integer myint = new Integer(5);
int builtinInt = myint.intValue() + 4;
myint = new Integer(builtinInt + 5);
```

05-22: Autoboxing

- Normally, you can just treat an Integer like an int
  - Autoboxing handles conversion for you
- There are some cases where this does cause a problem

```
Integer i1 = new Integer(5);
Integer i2 = new Integer(7);
Integer i3 = new Integer(7);
System.out.println(i1 < i2);
System.out.println(i1 == i3);
```