Application Programming

Week 3

Lecture 2

Deliverables

- Arraylists
- Java I/O

Discussion

- Abstract Class
- Superclass
- super

ArrayLists

- An ***ArrayList*** object is an array that can grow or shrink as needed.
  - In our Payroll System example we don’t know the number of employees that will be added to a company.
    - If we create an array for more than we have, we waste space.
    - If we try to add an employee past the end of the array, we get an exception!
  - Use an ArrayList when you don’t know how many of something you need.

ArrayList Methods

- Some of the methods:
    - to add an object to the ArrayList
        - `add(Object o)`
        - `add(int index, Object o)`
    - to get an object from the ArrayList
        - `get(int index)`
    - to tell you how many things are in the ArrayList
        - `size()`
    - to remove an object from an index in the ArrayList
        - `remove(int index)`

ArrayList Methods

- Look in the Java API for ArrayList
    - Open the package java.util.
    - Click on the class ArrayList.

- Also described in the textbook - Chapter 7

ArrayList Examples

- Example 1:

```
ArrayList grades = new ArrayList();
grades.add( 100 );
grades.add( 97 );
```

- Example 2:

```
ArrayList letters = new ArrayList();
letters.add( “A” );
letters.add( “B” );
```

ArrayList Examples

More on generics later :)

- Example 1:

```
ArrayList<Integer> grades = new ArrayList<Integer>();
grades.add( 100 );
grades.add( 97 );
```

- Example 2:

```
ArrayList<String> letters = new ArrayList<String>();
letters.add( “A” );
letters.add( “B” );
```

User I/O: The Console

- Input from the user - typing into the console → `System.in`

```
Scanner input;
input = new Scanner( System.in );
// Prompt user to enter input
int x = input.nextInt();
```

- Output to the user - printing to the console → `System.out`

```
System.out.println( “hi” );
System.out.print( “hello” );
System.out.printf( “%s”, “hey” );
```

File I/O - Scanner

- Input - reading from a file

```
// create a File object
File file = new File( “data.txt” );
// open the file for reading
Scanner scan = new Scanner( file );
// for each line or character…
while( scan.hasNext() ){
     // read it in & use it
     String token = scan.next();
}
// close the file
scan.close();
```

File I/O - FileWriter

- Output - printing to a file

```
File file = new File( “data.txt” );
FileWriter printer = new FileWriter( file );
printer.write( “line one” );
printer.write( “line two” );
printer.close();
```

Buffered Reading from a File

- To read from a character based file use a `FileReader` object with a `BufferedReader` object.
  - The `FileReader` class knows how to read character data from a file.
  - `BufferedReader` is used to buffer the data as you read it from the disk into memory.
    - disks are much slower to read from than memory
    - so read a big chunk from disk into memory
    - and then read from the chunk in memory as needed
    - this is known as buffering, so you don't have to wait

Buffered Writing to a File

- Very similar to reading from a file, but use `FileWriter` and `BufferedWriter`
- Write out things with the method
  - `write(string);`
- Force a new line with
  - `newLine();`
    - Different systems use different ways to end a line (Macs versus Windows)
    - This will write it out in away that works for the current system.

Example: Payroll System

- Back to the Company & Employees
  - Add an ArrayList to the Company class, it should contain Employee objects.
    - (...Where should it be initialized?)
  - Add all employees to a file
    - .csv files are “comma-separated” files, which can be opened as text or as a spreadsheet.
    - Modify the code to read from a file, to add Employee objects to the new ArrayList.