CMPE212 – Reminders

• You can now work on Exercise 4 on File I/O – this will not be covered in class.
• Assignment 1 due tomorrow.
• Quiz 1 Next Week.

Quiz 1 Topics

• Everything up to and including Monday’s lecture, next week.
• Exercises 1, 3 and 4; and assignment 1 are fair game and would be good preparation for the Quiz.
• More details on next slide:

Quiz 1 Topics, Cont.

• Emphasis on:
  – Java history and operation.
  – Primitive types, variable declaration and initialization, expressions, conditionals and loops.
  – Type casting of primitive types.
  – Screen (or "console") output using System.out.println() and System.out.printf().
  – Screen input using the Scanner class.
  – Arrays – one and two dimensional.
  – Methods and attributes – use, declaration and coding.
  – java.lang classes, including the String and wrapper classes.
  – Method Overloading?

Quiz 1 Restrictions

• You must write the quiz in the Thursday 8:30am lab. It will be open from 8:30 to 10:30, but make sure to start it before 9:30 to be sure you have enough time.
• The quiz can only be opened in JEFF155 or JEFF157 – it is IP address restricted to only the lab machines.

Quiz 1 Format

• Electronic – in onQ.
• ~45 minutes.
• T/F and short answer.
• Short answer could be a code segment, writing a complete method or possibly a complete program.
• Code writing will be towards the end of the quiz – leave yourself enough time!!!
• You will need to be able to read and write procedural Java code.

Quiz 1 Format, Cont.

• When you type code directly into a multi-line answer box, avoid using the <tab> key.
• This key will not give you a tabbed space but will just move the cursor out of the box, which is really annoying.
• Use just the spacebar.
• Style is not critical, but nice looking code is easier to read and grade by an old guy like me...
Quiz 1 Rules

- You will use onQ only and no other programs can be used on any device.
- No other aids.
- The TA must be able to see your screen.
- No talking to other students when writing. If you have written, please keep the noise down until all other students have finished writing the quiz.

Today

- Some useful classes from java.lang that we should cover before getting into OOP.

Some Useful Java Classes

- The classes defined in the java.lang package are automatically imported for you, since they are used quite often.
- They include:
  - The Wrapper classes
  - Math
  - Object
  - String
  - System
  - Thread
- (How do I know all this?...)

Aside – the Java API Documentation

- Application Programming Interface.
- This is the name of the huge collection of classes, or “libraries”, that support the operation of a Java program.
- The “API Docs” is the reference documentation that tells you how to use these classes.
- You can access the API docs on-line or download your own copy.
- Let’s have a look!

java.lang

- This is the only package (and all its interfaces, enums, classes, Exceptions, Errors and annotations) that is automatically imported into every Java program.
- If you need something that is in another package you will need to write a specific import statement for that class or for the entire package containing that class. Your IDE should be able to help you create the appropriate import statement – Eclipse does.

Aside - static Methods

- Many of these java.lang classes are utilitarian in nature – they contain many static methods:
  - static attributes and methods are loaded once into memory and not garbage collected until main is finished.
  - These methods will run faster the next time(s) they are invoked.
  - Generally, they are utility methods that do not depend on the values of a class’ attributes.
### static Methods, Cont.

- static methods can be invoked without instantiation of the Object that owns them. `Math.random()`, for example.
- static methods and attributes are shared by all instances of a class – there is only one copy of these methods in memory.
- A static method can only invoke other static methods in its own class – you can’t have pieces of code disappearing from a static method in memory...
- This is all done for reasons of ease of use and efficiency.

### Math Class

- As you would expect:
  - A collection of static constants and static mathematical methods.
  - You cannot instantiate the Math class, but why would you want to?
  - Let’s look over the API Docs.

### Wrapper Classes

- Sometimes it is necessary for a primitive type value to be an Object, rather than just a primitive type.
  - Some data structures only store Objects.
  - Some Java methods only work on Objects.
- Wrapper classes also contain some useful constants and a few handy methods.

### Wrapper Classes - Cont.

- Each primitive type has an associated **wrapper class**:
  - `char` Character
  - `int` Integer
  - `long` Long
  - `float` Float
  - `double` Double
- Each wrapper class `Object` can hold the value that would normally be contained in the primitive type variable, but now has a number of useful static methods.

### Wrapper Classes - Cont.

- Constructors are depreciated, so use:
  - `Integer number = Integer.valueOf(46);`
  - `Or:
  - Integer num = Integer.valueOf("56783");`
  - `Or:
  - Integer num2 = Integer.valueOf("10010010", 2);`
  - `valueOf` is a static “generator” method. This kind of thing is getting used more often in Java.

### Wrapper Classes - Cont.

- A few other things:
  - `Integer.MAX_VALUE // gives maximum integer`
  - `Integer.MIN_VALUE // gives minimum integer`
  - `Integer.parseInt("453") // returns 453`
  - `Integer.toString(653) // returns "653"`
  - `number.equals(num) // returns false`
  - `int aNumber = number.intValue(); // aNumber is 46`
Wrapper Classes – Cont.

• The `Double` wrapper class has equivalent methods:

  ```
  Double.MAX_VALUE // gives maximum double value
  Double.MIN_VALUE // gives minimum double value
  Double.parseDouble("0.45E-3") // returns 0.45E-3
  ```

• The parse... methods will throw a `NumberFormatException` if they cannot create the appropriate number.

• See the API Docs for other methods and constants dealing with NaN and -Infinity and Infinity.

System Class

• We have used (see Exercise 1):
  ```
  System.out.println()
  System.out.print()
  System.out.printf()
  ```

• Also:
  ```
  System.err.println()
  ```

Other Useful System Class Methods

• `System.currentTimeMillis()`
  ```
  Returns, as a long, the number of milliseconds elapsed since midnight Jan. 1, 1970.
  ```

• `System.exit(0)`
  ```
  Immediate termination of your program.
  ```

• `System.getProperties()`
  ```
  All kinds of system specific info - see the API.
  ```

• `System.getProperty(string)`
  ```
  Displays single system property.
  ```

• `System.nanoTime()`
  ```
  Time in nanoseconds
  ```

Other Useful System Class Methods, Cont.

• `System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
  ```
  "Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array."
  ```

Strings, so Far

• String literals:
  ```
  "Press <enter> to continue."
  ```

• String variable declaration:
  ```
  String testStuff;
  or:
  String testStuff = "A testing string."
  ```

• String concatenation ("addition"):
  ```
  String testStuff = "Hello";
  System.out.println(testStuff + " to me!");
  Would print the following to the console window:

  Hello to me!
  ```
Strings - Cont.

- Escape sequences in Strings:
  - These sequences can be used to put special characters into a String:
    - \" a double quote
    - \' a single quote
    - \ a backslash
    - \n a linefeed
    - \r a carriage return
    - \t a tab character

Strings, so Far - Cont.

- For example, the code:
  ```java
  System.out.println("Hello\nclass!");
  ```
  prints the following to the screen:

  Hello
class!

String Class - Cont.

- Since Strings are Objects they can have methods.
- String methods (67 of them!) include these non-static members:
  - length()
  - equals(OtherString)
  - equalsIgnoreCase(OtherString)
  - toLowerCase()
  - toUpperCase()
  - trim()
  - charAt(Position)
  - substring(Start)
  - substring(Start, End)

String Class - Cont.

- String.format(String format, Object... args)
  - A static String method that works like the .printf() method except that it returns a String instead of sending it to the console.

String Class - Cont.

- A few examples:
  ```java
  int i;
  boolean abool;
  String testStuff = "A testing string."
  i = testStuff.length(); // i is 17
  abool = testStuff.equals("A testing string."); // abool is false
  abool = testStuff.equalsIgnoreCase("A TESTING STRING."); // abool is true
  ```
**String Class - Cont.**

```java
char aChar;

aChar = testStuff.charAt(2); // aChar is 't'
i = testStuff.indexOf("test"); // i is 2
```

**Aside - More about String's**

- Is "Hello class" (a String literal) an Object?
  - Yup, "Hello class".length() would return 12

- Also, String's are immutable – meaning that they cannot be altered, only re-assigned.
- There are no methods that can alter characters inside a string while leaving the rest alone.
- Arrays are mutable, in contrast – any element can be changed.

**Aside - StringBuilder**

- Also in java.lang: a mutable character buffer that works mostly like a String.
- (Also, StringBuffer, if you need a thread-safe version that is less efficient than StringBuilder.)
- Works by putting characters in a buffer. The buffer usually has extra room, and can grow once it is filled up.
- Three constructors:
  - Takes a String object or literal.
  - Takes an int – which will become the size of the buffer – the "capacity".
  - An empty constructor.

**StringBuilder, Cont.**

- If you don't specify a buffer size, then the buffer will grow (in chunks of 16 characters) as needed. Use .capacity() to get the buffer size.
- Use .length() to get the number of occupied positions in the buffer.
- .append() can be used to append a String, a char, or another StringBuilder object to the end of the string on which it is invoked.
- .insert() does an insert and takes a character position as its first argument. Insert a char, a String, or another StringBuilder object.
- .setCharAt() changes a character at a given position.

**StringBuilder, Cont.**

- .delete() deletes a substring, .deleteCharAt() deletes a single character.
- More methods are listed in the API docs.
- Also, [] does not work (as it would for an array), and .equals() will not work.
- To compare for equality, use .toString() to extract a String and then use .equals() on the String objects instead.
- .compareTo() does work properly if you need to sort a collection of StringBuilder objects.
- See the demo program: StringBuilderDemo.java.

**Aside – More Exercises**

- Exercise 5 (Palindromes) is on strings.