**CMPE212 – Reminders**

- Assignment 1 is posted.
- Lab starts this week (Thursday 8:30am) in JEFF155. We also have JEFF157, if we need it. Meet your TAs! Handle questions on:
  - JDK, Eclipse installation?
  - Exercises 1 and 3?
  - Assn 1.
- Don’t forget to do “Quiz 0”.

**Today**

- Continue Basic Java Syntax:
  - Primitive types
  - String objects
  - Array declaration
  - Literal values
  - Using var
  - Constants
  - Type Casting

**Primitive Types in Java**

- Java primitive types:
  - `char`
  - `byte`
  - `short`
  - `int`
  - `long`
  - `float`
  - `double`
  - `boolean`

**Primitive Types?**

- What is a *primitive type* anyways?
- Every other possible type in Java is an Object.
- A variable declared as one of the types shown on the previous slide is not an Object.
- Why does Java have primitive types?
- Object definition often involves the data abstraction or encapsulation of one or more primitive type attributes.

**Integer Primitive Types**

- `byte, short, int, long`
- For `byte`, from -128 to 127, inclusive (1 byte).
- For `short`, from -32768 to 32767, inclusive (2 bytes).
- For `int`, from -2147483648 to 2147483647, inclusive (4 bytes).
- For `long`, from -9223372036854775808 to 9223372036854775807, inclusive (8 bytes).
- A “byte” is 8 bits, where a “bit” is either 1 or 0.

**Aside - Number Ranges**

- Where do these min and max numbers come from?
- Memory limitations and the system used by Java (two’s complement) to store numbers determines the actual numbers.
- The Wrapper classes can be used to provide the values - for example:
  ```java
  Integer.MAX_VALUE    // returns the value 2147483647
  ```
- More on Wrapper classes later!
### Real Primitive Types

- Also called “Floating Point” Types:
  - **float, double**

- For **float**, (4 bytes) roughly $\pm 1.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$ to 7 significant digits.
- For **double**, (8 bytes) roughly $\pm 4.9 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$ to 15 significant digits.

### Character Primitive Type

- **char**

- From ‘\u0000’ to ‘\uffff’ inclusive, that is, from 0 to 65535 (base 10) or 0 to ffff (base 16, or “hexadecimal”). A variable of the **char** type represents a Unicode character. Can also be represented as ‘a’ or ‘b’, etc.

- Java is “Unicode Aware” by default. You cannot have a character that is just an ASCII character. This means that a char value always needs 2 bytes of memory, not just 1.

### Boolean Primitive Type

- **boolean** is either **true** or **false**.

### Aside - String Objects

- **String**’s are not primitive data types, but are Objects.
- A **String** can be declared in the same way as a primitive type using the keyword: **String**.

  ```java
  String example = "I am a string!";
  ```

- This is the same as the following instantiation:

  ```java
  String example = new String("I am a string!");
  ```

### Text Blocks in Java 13

String multi = ""
A multiline string.
See, here’s another line!
Maybe a nice feature?"

- Good for formatted text like html, xml, json, sql queries and regex.
- Must have an empty line (or whitespace only) after the first "".
- Note the lack of an escape sequence for the single quote and how the indentation is removed.
- In Java 13, you have to run the compiler in “preview” mode to use text blocks.

### Array Declaration

- Declaration in Java:

  ```java
  int[] data = new int[10];
  ```

- **new** is always involved with the **instantiation** of an Object.
- Arrays are Objects in Java.
Array Declaration, Cont.

- An array literal is just a set of same-type values separated by commas in a set of { }.
- Only useful for small arrays...
- To get the size of an array, use the .length attribute.
- For example data.length would provide 10 for the array on the previous slide.
- Array indices range from 0 to .length - 1.

Array Declaration, Cont.

- Cannot dynamically size an array in Java (nothing like malloc() in Java)
- In Java, you cannot use pointer arithmetic to access array elements, only the indices.
- (You can never obtain a memory address in Java!)
- The first array element is at index zero.
- In Java you will get an immediate error and your program will halt if you try to go beyond the bounds of your array.

A Common Runtime Error

```java
public static void main(String[] args) {
    int[] nums = {10, 20, 4, -2, 6, 7, 8, 12};
    int i = 0;
    while (i <= nums.length) {
        System.out.println(nums[i]);
        i++;
    } // end while
} // end main
```

- Why would this fail?

A Common Runtime Error, Cont.

```java
public static void main(String[] args) {
    int[] nums = {10, 20, 4, -2, 6, 7, 8, 12};
    int i = 0;
    while (i <= nums.length) {
        System.out.println(nums[i]);
        i++;
    } // end while
} // end main
```

- Console output:

```
10
20
4
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 8
at HelloWorld.main(HelloWorld.java:11)
```

- What's in the error message?
- Why does it appear before the error happens?

Literal Values

- Integers, for example:
  - 12
  - -142
  - 0
  - 33344891

- If you write these kinds of numbers into your program, Java will assume them to be of type int, and store them accordingly.
- If you want them to be of type long, then you must append a "L" to the number:
  - 43L
  - 9999983475L
  - -2233487L

Binary, Octal and Hex Literals

- Use the prefix 0b (or 0B) in front of the numbers to get a binary literal.
- Octal – just use 0
- Hex use 0x
- You can also use the underscore _ in-between digits, to aid in visualizing the number.
- System.out.println(), by default displays the numbers in base 10.
- (Use printf to show the numbers in another codex or base. See part 2 of Exercise 1.)
Binary, Octal and Hex Literals, Cont.

- Summary examples of other literals:
  - binary: 0b111010111
  - binary with underscores: 0b111_010_111
  - octal: 07321023
  - hex: 0xab10fc

Literal Values - Cont.

- Real or “Floating Point” numbers, for example:
  4.5 -1.0 3.457E-10 -3.4E45

- These literals will be assumed to be of the “double” type.
- If you want them to be stored as ‘float’ types, append an “F”:
  3.456F 5.678E-10F -321.0F

Literal Values - Cont.

- char literals: ‘A’ ‘5’ ‘\u0032’
- boolean literals:
  true false
- String literals:
  "Hello there!"  "456.7"  "West of North"
  And, using "" to create a text block in Java 13.

Legal Variable Names

- Java names may contain any number of letters, numbers and underscore (“_”) characters, but they must begin with a letter.
- Standard Java Naming Conventions:
  - Names beginning with lowercase letters are variables or methods.
  - Names beginning with uppercase letters are class names.
  - Successive words within a name are capitalized (“CamelCase”).
  - Names in all capital letters are constants.

Local Variable Declaration

- Carried out inside a method.
- Syntax:
  type variable_name [=value];

Variable Declaration - Cont.

- Java may prevent you from using variables that are not initialized.
- So, it is often good practice to initialize your variables before use, for example:
  
  ```java
  int numDaysInYear = 365;
  double avgNumDaysInYear = 365.25;
  String greetingLine = "Hello there!";
  long counter = 0;
  ```
Java 10 var Type

- var is not a Java keyword instead it is called a "reserved type name".
- This allows code like:
  ```java
  var aNum = 10;
  ```
- aNum will be of type int.
- The type for the variable is inferred from the type of the literal value used to initialize the variable.

Java 10 var Type, Cont.

- This uses "type inference", which will only work if the type can be figured out from the same line of code as the declaration.
- So, variables have to be initialized using a literal or the result of some expression or method call.
- They can only be used for local variables, and:
  - You can also use them in for each loops (later), for the type of the collection element produced by iterating through the collection.

Java 10 var Type, Cont.

- What’s the point?
- var can reduce repetition in longer declarations. For example:
  ```java
  var collect = new ArrayList<String>();
  ```
- is easier than:
  ```java
  ArrayList<String> collect = new ArrayList<String>();
  ```
- and does not contain so much redundancy.

Java 10 var Type, Cont.

- Suppose you have a series of local variable declarations that don’t line up very nicely:
  ```java
  int aNum = 45;
  ArrayList<Integer> nums = new ArrayList<Integer>();
  String aString = “Hello Class!”;  
  ```
- You could write this as:
  ```java
  var aNum = 45;
  var nums = new ArrayList<Integer>();
  var aString = “Hello Class!”;
  ```

Constant Attribute Declaration

- Syntax:
  ```java
  [private|public] [static] final type ATTRIBUTE_NAME = literal_value;
  ```
- The Java keyword, final can be used to make sure a variable value is no longer “variable”.
- Usually these are declared public static.
- The value must be assigned – this part is no longer optional.

Constants, Cont.

- Java will not allow your program to change a constant’s value once it has been declared.
- For example:
  ```java
  final int NOM_DAYS_IN_YEAR = 365;
  final double MM_PER_INCH = 25.4;
  ```
- Note that constant names are all in upper-case, by convention.
- You can also declare constants inside a method.
Type Casting

- When a value of one type is stored into a variable of another type.
- Casting of primitive types in one direction is automatic, you do not have to deliberately or “explicitly” cast:
  
  byte > short > int > long > float > double

Type Casting - Cont.

- For example in the statement:

  ```java
double myVar = 3;
``` 

  the number 3 is automatically cast to a double (3.0) before it is stored in the variable “myVar”.

- However, if you tried the following:

  ```java
  int anotherVar = 345.892;
  ```

  the compiler would protest loudly because a double cannot be stored in an int variable without loss of precision. Wrong direction!

Casting Operator

- If you really want to cast in the other direction, then you must make an explicit cast. For example:

  ```java
  int anotherVar = (int)345.892;
  ```

  is legal. The "(int)" part of the statement casts the double to an int. The variable anotherVar would hold the value 345

- Note how numbers are truncated, not rounded!

Aside – Casting Objects

- We will find (later) that you can cast objects using the casting operator too.

- Objects must have an inheritance relationship in order for the cast to succeed.

- For example:

  ```java
  Integer aVal = new Integer(45); 
  Number aNum = (Number)aVal;
  ```

- The Integer class extends the Number class.

- More about this later...