CMPE212 – Reminders

• Course Web Site:
  http://research.cs.queensu.ca/home/cmpe212

• Lab starts next week 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? (If it is posted soon…)

• Don’t forget to do “Quiz 0”.

CMPE212 – Today

• Start Java Syntax:
  – Simple class structure.
  – Attribute declaration.
  – Method declaration.
  – Access modifiers.
  – static keyword (first pass)
  – return statement
  – Some simple examples
  – Primitive types

• Compare Java and C. So far…

OOP in Java

• A class or “object definition” or an “object”, consists of instance variables and/or methods.
• By convention, instance variables are all declared before the methods:

```java
public class ShowStructure {
    // instance variables or “attributes” here
    // methods here
}
```

OOP in Java – Cont.

• In Java, a class is an Object, and an Object is a class (rather Zen is it not!) And, you can extend this by saying that an instance of a class is also an Object.

• Code and attributes cannot be defined outside of a class.
• The only code that can exist outside a method are attribute declarations or other (“inner” or “nested”) class definitions.

Aside – Class Instances

• More on this topic and Objects, later. But, for now:
  • static class members can be used directly, with creating a instance of a class.
  • Otherwise a class cannot be used directly without first creating a copy or an “instance” of that class.
  • The class definition serves as a “blueprint” and the new keyword is used to create an instance of that blueprint.
  • Constructors can be used to initialize attributes at the time of instantiation.

Attributes

• Also called “class variables” or “instance variables” or “fields”.
• Declared within a class at the same level as the method declarations.
• These variables are known to all methods within a class (their “scope”).
• You can control their privacy and the way they are stored in memory (using public/private/protected and static).
Aside – Access Modifiers

- **public** means the attribute or method is available to any external class (as well as inside the class).

- **private** means that the attribute or method, the “member”, is only available inside the class in which it is declared.

- **protected** means the member is only public to classes in the same package as the class in which the member is declared.

- Public or protected access must still be carried out through the class or the instance of the class as the owner of the member.

Access Modifiers, Cont.

- Inside a class, the access modifier does not have any effect.

- All members of a class are available to all other members in that class, regardless of the access modifier.

static (First Pass)

- **static** means different things depending on where it is used. For now, consider:

  - public static members are available for use outside the class without the need to instantiate the class.

  - Any static member remains in memory until the program is complete.

  - Since main is static, it can only invoke other static methods when they are in the same class.

Aside - Packages

- More on this topic later too!

  - A package is a folder inside the src folder. You associate a class with a package using the package keyword at the top of the class.

  - Organizes code in large projects and makes it easier to import it.

Attribute Declaration

- Syntax:

  ```java
  [private|public] [static] [final] type attributeName [= literal];
  ```

  - Note that the type part is not optional – this is why java is a declarative language.

  - And, a variable cannot change its type later, called static typing.

  - You cannot use a variable unless you have declared it first.

Variable Declaration

- Declaring a variable inside a method gives that variable the scope of just inside the method, not outside the method.

  - Generally, a variable is only available inside the block {…} in which it is declared.

  - The syntax for declaration inside a method is the same except you don’t need the [private|public] [static] [final] parts.

  - Also, in Java 10 and later, you can type a local variable as var. The type will be determined at run-time by context (More on this later.)
Method Declaration (a “Header”)

- The syntax for simple method declaration:

  ```java
  [private|public] [static] [final] returnType methodName
  ([parameterList]) {…}
  ```

- If `main` invokes methods or uses attributes in the same class as itself then those attributes and methods must also be declared `static`.

Method Declaration - Cont.

- A method must have a `returnType`.
- The `returnType` can be any single Object or a primitive type. (For example: an `int`, `double`, `String`, an array (like `int[]`), or any other pre-defined Object.)
- If the method does not return anything, then the keyword `void` is used instead.
- The `main` method does not return any value, so that’s why it is declared as in:

  ```java
  public static void main (String[] args) {…}
  ```

Aside - The `main` Method

- For the JVM to run an application, it must know where to start.
- By design, the starting point is always the execution of the `main` method.
- The JVM expects the `main` method to be declared exactly as shown – the only thing you can change is the name of the String array, called `args` above.

Method Declaration - Cont.

- `parameterList` provides a means of passing items, or parameters, into a method.
- It is optional.
- It can consist of one or many parameters, separated by commas.
- Each parameter type must be declared in the parameter list, as in `type variableName, type variableName, ...`
- Java does not have default arguments as does Python and C++.

Aside - varargs

- You can use the ellipsis operator (three dots - ...) to create a parameter that can take any number of arguments of that type. For example, suppose you have an object of type `Bling`:

  ```java
  public void seeBling (Bling... blingers) {// code }
  ```

- Within `seeBling`, you get at the individual arguments of type `Bling` by pretending that `blingers` is an array (use `[]` along with an index value.)

Aside – varargs, Cont.

- It is not used too often, but,
- See the declaration of the `printf` method for example:

  ```java
  public PrintStream printf(String format, Object... args)
  ```
Methods - the return Statement

- Unless the return type is void, the method must contain at least one return statement. A void method can also use a return statement without anything after it, as a means of terminating the method.
- A method always stops (terminates) when a return statement is encountered.
- Syntax:

  ```java
  return [literal|expression];
  ```

- The type of literal|expression must match the return type specified in the method declaration statement.

Method Examples

```java
public void printHello() {
    System.out.println("Hello");
} // end printHello

public void printHelloName(String yourName) {
    System.out.println("Hello " + yourName);
} // end printHelloName

public void printAvg(int a, int b) {
    System.out.println((a + b) / 2.0);
} // end printAvg
```

Method Examples - Cont.

```java
public double average(double a, double b) {
    return (a + b) / 2;
} // end average

public int lowest(int a, int b) {
    if (a <= b)
        return a;
    else
        return b;
} // end lowest
```

Attribute Examples

```java
public double aVar;

public int aNum = 100;

private String hello = "Hello";

var aVal = 4.5E-10; // aVal will be of type double based on context - the literal being assigned is a double type literal.
```

static Again

- If static methods are using attributes in the same class, then those attributes must also be static.
- Don’t worry the pre-compiler will also remind you!

Example: A Simple Class

```java
public class Simple {
    public static int aNum = 100;

    public static int sumNums(int num1, int num2) {
        return num1 + num2;
    }

    public static void main(String[] args) {
        int anotherNum = 200;
        System.out.println(sumNums(aNum, anotherNum));
    }
}
```
Another Simple Class

// An example of the use of varargs (the ellipse...)
// Also a for each loop!
public class AnotherSimple {
    public static int sumNums(int... nums) {
        int sum = 0;
        for (int num : nums)
            sum += num;
        return sum;
    }
    public static void main(String[] args) {
        System.out.println(sumNums(2, 5, 7, 10, 3));
    }
} // end AnotherSimple class

Aside – static Methods

• main can invoke static methods in the same class without naming its class as the owner.
• Since these are public static methods we could access them from outside the class as in:
  System.out.println(Simple.sumNums(4, 5));
  System.out.println(Simple.aNum);

Java versus C

• What is the main difference between C and Java?
• Java code must be contained within an Object. Creating a program is now all about defining Objects (or classes).
• In Java, you cannot have any code outside a class definition.
• The only code you can have outside a method definition are attribute declarations.

Java versus C, Cont.

• C has #include, Java has import:
  import java.io.Serializable;
• C has function prototypes, Java does not.
• C has global variables, Java does not.
• C has functions, Java has methods (what is the difference?)
  public static void printCodes() {
• C has a main() function, Java has a main method:
  public static void main(String[] args) {

Java versus C, Cont.

• Some stuff is the same:
  • { } used to delineate methods.
  • ; at end of line.
  • /* for in-line comments.
  • String literal enclosed in " ".
  • ( ) contain arguments and parameter lists and can be used to establish precedence in expressions.
  • Space used to delineate elements of expressions.
Primitive Types in Java

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

Primitive Types?

• What is a **primitive type** anyways?
  - Everything else 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 construction 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:
    - `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 ±1.4 x 10^-38 to ±3.4 x 10^38 to 7 significant digits.
• For *double*, (8 bytes) roughly ±4.9 x 10^-308 to ±1.7 x 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 '8', 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.