WIT COMP1000

Variable Scope

Variable Scope

- All variables have a set *scope*
  - » Parts of the code where that variable can be used
- Variables declared in a method are *local variables* for that method
  - » Can not be used outside of that method, i.e., can not be used in other methods
- Method parameter variables are treated as local variables in that method

Example

```
import java.util.Scanner;

public class ClassExamples {

    public static void main(String[] args) {
        @SuppressWarnings("resource")
        Scanner input = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int input_value = input.nextInt();

        int result = factorial(input_value);
        System.out.println(input_value + "!=" + result);
    }

    public static int factorial(int n) {
        int total = 1;
        while (n > 0) {
            total = total * n;
            n--;
        }
        return total;
    }

}
```

input is local to the `main()` method

input_value is local to the `main()` method

result is local to the `main()` method

n is local to the `factorial()` method

total is local to the `factorial()` method

Different Scopes == Different Variables

- Variables in different scopes can have the same name (and be different types)
- They are different variables!
- Two variables with the same name but in different scopes are *not related in any way*
- To avoid confusion, do not reuse variable names in different methods or scopes

Poor Example: Don't Do This!

```
public class ClassExamples {

      public static void main(String[] args) {
            double my_num = 10.5;
            double res;
            System.out.println("main(): my_num=" + my_num);
            res = myMethod();
            System.out.println("main(): my_num=" + my_num);
            System.out.println("main(): res=" + res);
      }

      public static double myMethod() {
            double my_num = 75.32;
            System.out.println("myMethod(): my_num=" + my_
            return my_num;
      }

}
```

my_num is local to the main() method

my_num is local to the my_method() method

Class Scope Variables

- Variables and constants can be placed in the *class* scope by declaring them outside of all methods, but still inside the { } for the class
  - » We'll look at some simple examples now and talk more about this in detail later
  - » For now, most often useful for *constants* that are used in multiple methods
- Variables and constants can not be placed outside of the class

Constants

- It’s usually a good idea to name constants in your program if they have some special meaning
- By convention, variables names with all capital letters are constants
- Java includes `final` "variables" to strictly enforce the idea of a constant (value can not be changed after initialization)
  - » Example: `static final int CENTS_PER_DOLLAR = 100;`
  - » Generic form: `static final TYPE NAME = VALUE;`
- We'll talk more about the meaning of `static` later

Example with a Class Scope Constant

```
public class ClassExamples {

    static final double DOLLARS_PER_EURO = 1.14;

    public static void main(String[] args) {
        System.out.printf("5 dollars is %.2f euros%n", dollarsToEuros(5));
        System.out.printf("5 euros is %.2f dollars%n", eurosToDollars(5));
    }

    public static double dollarsToEuros(double dollars) {
        return dollars / DOLLARS_PER_EURO;
    }

    public static double eurosToDollars(double euros) {
        return euros * DOLLARS_PER_EURO;
    }
}
```

Exercise

- Write a program that uses the famous `E = mc`^{2} formula to calculate mass and energy equivalence in both directions
  - » Use a class scope constant for the value of `c` (299792458 m/s)
  - » Write a method that calculates the energy given a set amount of mass
  - » Write a method that calculates the mass given a set amount of energy
  - » Write a `main()` method to test each other method

Answer

```
public class ClassExamples {

    // meters/sec
    static final int C = 299792458;

    public static void main(String[] args) {

        System.out.printf("1 kilogram = %.3f joules%n", energyFromMass(1));
        System.out.printf("1000000000 joules = %.9f kilograms%n", massFromEnergy(1000000000));

    }

    public static double energyFromMass(double mass) {
        return mass * C * C;
    }

    public static double massFromEnergy(double energy) {
        return energy / (C * C);
    }
}
```

Class Scope Variable Gotcha

- If you have a class scope variable and a local variable in a method with the same name, the local variable "hides" the class scope variable
- The two variables are declared in different scopes, so they are completely different variables
- The class scope variable will not be accessible within the same scope as a local variable that has the same name
    - » Another reason not to use class scope variables for now!

Poor Example: Don't Do This!

```
public class ClassExamples {

    static int my_var = 10;

    publi  static void main(String[] args)
        i    my_var = 42;
        my    hod();
        Sy     .out.println("main(): my_var=
    }

    public        void myMethod() {
        Syst          rintln("myMethod(): my_var=" + my_var);
    }
}
```

my_var is a class variable and would be accessible in all methods

my_var is redeclared within the main() method here, so any uses of my_var in main() will use the local variable, not the class one

if you do use a class scope variable, make it a constant with **final** (and don't't' reuse the name!)

this use of my_var is not in the main() method, so it will use the class variable

Other Scope Rules

- Any variables declared within a code block (everything between a set of braces `{}`), are local to that block
- Variables declared inside of an **`if`**-**`else`** block, **`while`** loop, or **`for`** loop can only be used inside of that block or loop
- Similar rules apply for "hiding" variables of the same name from an outer scope as with class scope variables
  - » One more time: don't reuse variable names!

Example

```
public class ClassExamples {
    public static void main(String[] args) {
        int i;
        for (i = 0; i < 10; i++) {
            int j;
            j = i * 9;
            System.out.println(j);
        }
        System.out.println(i);
    }
}
```

i can be used anywhere in the main() method

j can only be used in the for loop body

Example with an Error

```
public class ClassExamples {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println(i);
        }
        System.out.println(i);
    }
}
```

`i` can only be used in the **for** loop

Error! `i` can't be used outside of the **for** loop

Take Home Points

- All variables and constants have a certain scope (class, method, block)
- Variables can only be used within the same scope or any sub-scopes
- Be very careful about reusing variable names
- Class constants are useful, but class variables should only be used in certain cases which we'll discuss in detail later