Programming 1

Week 05 – Type of Errors

Type of Errors

- When you write programs, things don't always work correctly. In fact, one of the reasons why we need to test our programs - we want to make sure they work correctly. Three types of errors:
  - Syntax errors (i.e., grammar)
  - Logical errors (“bugs”) are the hardest to fix
  - Run-time error messages

Syntax errors

- Syntax errors are mistakes that the programmer has made that violate the rules of the programming language. These errors must be corrected before the compiler can translate the source code. For example:
  - Misspellings, mismatched parentheses
  - Forgetting to write a colon at the end of a case statement.
  - Calling a method with wrong number or wrong type of arguments

The following list describes some errors that are easy to make in writing Java …

- Class name does not match file name. (usually this is due to uppercase vs lower case mistake or simple typo)
- misspelled variable name. (use of variable name does not match name in its declaration)
- missing semicolon after assignment statement, variable definition, import statement
- import statement naming package not classes
- missing parenthesis "**(**" and "**)**" around condition in if or while statement
- missing the variable type for an argument in the parameter list of method declaration
- redefining the type of a variable (defining a variable which is already defined)
- confusing numeric char '2' with int 2
- Missing } brackets: This is a common programming error in many programming languages and can be fixed by adding the closing bracket. All blocks of code must be started with a { bracket and ended/closed with a }bracket. This type of error can be reduced by using a good code indentation system.
- Missing) brackets. This is a common programming error in many programming languages and can be fixed by adding the closing bracket.

Logical errors

- Logical errors are mistakes that cause the program to produce erroneous results. Once a model of the program is assembled, it should be checked for these errors.
- You can provide sample data and predict what the output should be. If the program does not produce the correct output, a logical error is present in the program.
  - No error message; you have to recognize that the answer (or behavior) is wrong. For Example:
  - + where you meant *, <= where you meant <, wrong order…
  - Forgetting to write a break statement in a case section.

Runtime error

- A runtime error is an error that occurs while the program is running. These are usually logical errors, such as mathematical mistakes.
- Testing for runtime errors requires that the program be executed with sample data or sample input.
  - division by zero

Examples..

- "Cannot find symbol" errors generally occur when you try to reference an undeclared variable in your code. Consider the following example:

```
public class Test {
    public static void main(String[] args) {
        int a = 3;
        int b = 4;
        int c = 20;

        average = (a + b + c)/5.0;
        System.out.println(average);
    }
}
```

Here, the variable `average` has not been declared — you need to tell the compiler what the type of average is: for example: `double average = (a + b + c)/5.0;`

Examples..

- This error could also occur if you forget to import a Java package that you need to use. For example, consider the following program that reads in an integer from the user:

```
public class Test {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        int n = console.nextInt();
    }
}
```

The issue here is that the program must import `java.util.Scanner` (or, more generally, `java.util.*`). Otherwise, the compiler does not know what a `Scanner` type is.

Examples..

- `incompatible types.` This error occurs when there are type issues with your program. It is possible to convert between some kinds of types; for example, you can freely convert a `char` to an `int` and vice versa, and you can also convert a `double` to an `int` with some typecasting. However, you cannot convert between primitive types and objects such as `String`. For example:

```
public class Test {
    public static void main(String[] args) {
        int num = "Hello, world!";
    }
}
```

Examples..

- `variable might not have been initialized.` This error occurs when the compiler believes you're trying to use a variable that has not been "initialized" — or given an initial value — yet. In a very simple case:

```
public class Test {
    public static void main(String[] args) {
        int x = 2;
        int y;
        System.out.println(x + y);
    }
}
```

Here, you have not told the compiler what the value of y is. Therefore, y cannot be printed; it needs to be initialized as x is in this example.

Examples..

- In this example, the programmer intends to calculate the sum of two numbers, num1 and num2. However, due to a logical error, the sum of num1 and num1 is calculated instead.

```
public class Test {
    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 20;
        int sum = num1 + num1;
        System.out.printf("Sum: %d", sum);    }
}
```

Syntax is correct, but logic is incorrect

Examples..

- Missing the "main" method. All java applications must have a main( ) method that has the following form ...

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

}
```

- If you mistype any part of this line or miss out a keyword, then a run-time error will be generated.
- For example, if you omit the keyword `static` then an error message of the form:
- `Exception in thread main.....`will be generated at run time.

Examples..

- In more complicated scenarios, `if` statements can cause this error if you are not careful about ensuring that a variable is initialized. For example:

```
public static void main(String[] args) {
        int x;
        boolean setX = false;
        if (setX) {
            x = 10;
        }
        System.out.println(x);
    }
}
```

Examples..

- Error in the Mathematic order of operations and it is a common programming error to omit brackets when doing math opeartions. For example:

  x = a * b + c;

- may result in a different value stored in x then would ...

x = a * (b + c);

Guidelines for Correcting Errors

- Taking the time to write a correct and well-documented program from the start of the programming process will save you time in the long run. Finding and correcting program "bugs" (logic errors) takes a lot of time, and it can be frustrating, especially if you neglected to include code comments
- Debug on a small scale. Thoroughly test each component of an application (each method of each object) piece by piece, correcting errors as you go. This is much easier than finding errors in a large program.

Guidelines for Correcting Errors

- Understand the symptom. Before changing a program, understand why the error occurred and where in the program it may have occurred. Look at what the program tried to do and where it went wrong.
- Use problem solving techniques to try to determine why an error happened how it could have arisen.
- Find the source of the error (find the ***root cause***). Don't waste time making "random" changes to your program.
- Correct the errors and test your program thoroughly using various test cases.

Hands-on

- Write a program that calculates the take-home pay for an employee. The two types of employees are salaried and hourly. Allow the user to input the employee first and last name, id, and type. If an employee is salaried, allow the user to input the salary amount. If an employee is hourly, allow the user to input the hourly rate and the number of hours clocked for the week. For hourly employees, overtime is paid for hours over 40 at a rate of 1.5 of the base rate. For all employees' take-home pay, federal tax of 10% is deducted. A retirement contribution of 10% and a Social Security tax rate of 6% should also be deducted. Use appropriate constants.