INFO 5100
Application Engineering and Development

Agenda

- Background of Java
- Write/Compile/Run a program in Java
- Types
- Function
- Control Flow
- Class and Object

Background of Java

- Java promise to: Write once, run everywhere
- Java is a compiled, strong typed, imperative, object-oriented programming language
- Java language provides **JDK** (Java Developer Kit) which includes awesome functionalities provided by Java developers
- Java code eventually run on an environment with **JRE** (Java Runtime Environment) and runs on top of JVM (Java Virtual Machine)

First Program

```java
public class HelloWorld {
   public static void main(String[] args) {
       System.out.println("welcome to info 5100");
       String output = "hello world";
       System.out.println(output);
   }
}
```

Write Java Program

- Java code are built with **Classes**
- Java source codes are organized in files with .java extension
- Java file name should reflect the public class name in the file
- A whole Java program has one entry point, the **static void main** function
- Each line of code need to end in ;

Variable

- Variable can be used to replace literal values, makes code more flexible
- When declaring variable we must associate type with it

Naming Convention

- Java community follows camel casing as naming convention
- Class naming
  - HelloWorld
  - HTTPServerConnection
  - NortheasternUniversity
- Variable and function naming
  - output
  - coolFunctionThatDoesStuff

Compile Java Program

- Use javac command to compile Java source code into Java bytecode
- Developer codes in Java syntax
- javac command compiles the code into bytecode
- java command runs the byte code on top of JVM
- During the compilation process, java optimizes your code

Second Program

```
public class Operations
{
    public static void main(String[] args)
    {
        System.out.println(100 - 3);
        System.out.println("2 * 3");
        System.out.println(5 / 2);
        System.out.println("4 + 6 = " + (4 + 6));
    }
}
```

Types in Java

Primitives

```
byte - 8 bit signed
short - 16 bit signed
int - 32 bit signed
long - 64 bit signed
float - 32 bit
double - 64 bit
char - 16 bit
boolean - true/false
```

Objects

```
String
HashMap
AwesomeClass
```

Types in Java

- Java is a strongly typed language
- Types can be classified into primitive and object types
- Primitive type need to start with lower cases, object type need to start with upper cases
- Primitive type is pre-defined by Java itself, other types are all object types

Types in Java

- Type of the variable and the value must match*
- In some cases, Java performs auto type casting
    - Numeric values might loss precision
    - Sometimes during operations

Zero Values

- In Java, if you declare a variable with a type without assigning it a value. Those variables will have zero values
- Primitive types’ zero values are
    - int: 0
    - bool: false
    - ...
- Object types’ zero value is **null**

Typed language vs Untyped language

- Strong typed:
  - C/C++
  - Java
  - Golang
- Untyped:
  - JavaScript
  - Ruby
  - Python
- Strong typed language normally more verbos, good for team collaboration and maintenance of the code.
- Untyped language speeds up development, but maintenance is hard
- Strong typed language generally have better performance since optimization can be performed during compile time and runtime

Function

```
public int multipleTwoNumbers(int a, int b) throws IOException {
       return a * b;
}
```

Function

- Access modifier
    - public
    - private
    - protected
    - default
- Static/Non Static
- Return type
- Function name
- Parameter list
- Optional exception
- Use return statement to return value

Control Flow

- Control flow adds dynamic into your program
- Branching flow
  - If else statement
  - Switch statement
- Looping flow
  - For loop statement
  - While loop statement

Branch Flow - if else

- if else if else is a common branch control flow statement
- Uses evaluated boolean statements to decide which code block to go to
- Subject to logic short circuit

Branching Flow - if else

```
public char getGrade(int score) {
        if (score >= 90) {
            return 'A';
        } else if (score >= 80) {
            return 'B';
        } else if (score >= 70) {
            return 'C';
        } else {
            return 'F'
        }
}
```

Branch Flow - switch

- Switch case is similar to if else, based on certain value matching, execute code block
- Switch cases are evaluated from top to bottom, until return statement or break statement is reached

```java
public String getLanguageFromCountry(String country) {
       String result;
       switch (country) {
           case "Japan":
               return "Japaneses";
           case "China":
               return "Chineses";
           case "US":
               result = "English";
           case "France":
               result = "French";
               break;
           default:
               return "alian";
       }
       return result;
 }
```

Loop Flow - while

```
public void printNumber(int size) {
       while (size > 0) {
           System.out.println(size);
           size--;
       }
}
```

Loop Flow - for

```
public static void printMoreNumber() {
       for (int i = 0; i <= 10; i++) {
           if (i == 4) {
               continue;
           }
           i++;
           System.out.println(i);
       }
}
```

Class

- Classes are building blocks of a bigger Java program
- Object oriented programming foundation
- Class consists of
  - Instance members
  - Class members
- Class can have
  - Variable member
  - Method member
- Member of the class can be invoked using . operator

```java
public class Dog {

      public String name;

      public Dog(String name) {
          this.name = name;
      }

      public String getName() {
          return this.name;
      }

      public void bark() {
          System.out.println("my name is: " + this.name);
      }

}
```

Object

- Objects are instances of classes
- It has the capability of defined by the classes

Object

```java
Dog d = new Dog("james");
System.out.println(d.getName());
d.bark();
```