### CISC124 – Java Syntax

#### Structure
- Basics of class structure
- Attribute declaration
- Method declaration
- Access modifiers
- Arrays (review)
- Strings

#### Flow control
- Conditional branching
- Expressions
- Example

---

### Class Structure

**Summary review:**
- A “class” is a template for creating instances of “objects”
- An object has state (saved in attributes) and behaviour (can compute something - a.k.a., code execution)
- A named class itself is an object of type `Class`
- A class contains members: attributes (“instance” or “class” variables, or “fields”) and methods (“functions”)
- Attributes and methods cannot exist outside a class
- Java code exists in methods
- Only code that exists outside methods is attribute definitions (and “inner” class definitions)

---

### Class Structure

```java
public class MyClassName {
    int number1 = 10;
    String name = "John";

    public static void main(String[] args) {
        // Method's body
    }

    private int doWork(int inNumber) {
        // Method's body
    }
}
```

---

### Attribute Declaration (Static)

- Attributes are declared at the same level as methods (good practice → declare them at beginning of the class)
- You control their visibility: public, private, protected
- And the way they are stored in memory: static

  - `public static` → attribute available outside the class without the need to instantiate the class
  - `static` → “type” is compulsory

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

  **Example:** Define a constant attribute:
  ```java
  public static final float pi = 3.1415;
  ```

---

### Method Declaration

- Private, public, static, final → have the same meanings as for attributes
- A method must return a value of `returnType`, unless the `void` modifier is specified.
- `main` → does not return any value and it is the program’s starting point:
  ```java
  public static void main(String[] args) {
      // Method's body
  }
  ```

---

F. de la Parra
Method Declaration - Body

```
public int power2(int inNum)
{ // Method’s body
   // LOCAL VARIABLES: exist only when
   // method is executing.
   int temp = inNum * inNum;
   // More statements terminated by “;”
   return temp; // Return value in temp
}
```

Method Invocation

- Assume that method `power2` is a member of class `SimpleMath`
- We use the “dot operator” to invoke `power2` from an external
  class `otherClass`, after we have created an object `mathObj`, of
  type `SimpleMath`:
  ```java
  SimpleMath mathObj = new SimpleMath();
  intVar = mathObj.power2(5);
  ```
- From inside class `SimpleMath`, we call `power2`
  just by name:
  ```java
  intVar2 = power2(6);
  ```

Primitive Types

- **Integer types**
  - `byte`
  - `short`
  - `int`
  - `long`
- **Floating-point types**
  - `float`
  - `double`

Value stored in memory location

- **Logical type**
  - `boolean`
- **Character type**
  - `char`

Class Types

- Classes are used as types ("templates") of instantiated objects
- Classes that we declare can be used as types of attributes and
  methods
- Classes in the Java libraries can also be used as types
- Example: strings are objects in Java. Instances of class `String`
  ```java
  String name = “John”;
  ```
  ```java
  String name = new String(“John”);
  ```
Arrays

- List of "fixed size" containing elements of the "same type"
- Array size is set at declaration time. Attribute length contains the number of elements (locations) in the array.
- Locations in the array are indexed \(0, (\text{length} - 1)\)

```java
int[] skillLevel = {1,2,3,4,5,6,7,8,9,10};
int[] skillLevel = new int[10];
skillLevel[0] = 1;
skillLevel[1] = 2;
```

Conditional Branching

```java
if (BooleanExpression)
    {Block1 of Statements}
else
    {Block2 of statements}
```

Conditional Branching – if-else ladder

```java
if (BooleanExpression1)
    {Block 1 of Statements}
else if (BooleanExpression2)
    {Block 2 of statements}
else if (BooleanExpression3)
    {Block 3 of statements}
    ...
else
    {Block n of statements}
```