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 contain 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

```
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
        }
}
```

ATTRIBUTES

METHOD

METHOD

CLASS DECL.

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` → attribute is available to any other external class

`private` → attribute is available inside the class

`protected` → attribute is available to classes in the same package

Attribute Declaration

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

- `Public, private, static` → attribute remains in memory until the program is completed
- `public static` → attribute available outside the class without the need to instantiate the class
- `static` → "type" is compulsory

Example: Define a constant attribute:

```
public static final float pi = 3.1415;
```


Method Declaration

```
[private|public] [static] [void][final]
returnType methodName ([parameterList])  {
method's body }
```

- `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 :

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

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`:

```
SimpleMath mathObj = new SimpleMath();
intVar = mathObj.power2(5);
```

- From inside class SimpleMath, we call power2 just by name:

```
intVar2 = power2(6);
```


Primitive Types

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

- Logical type
  - boolean
- Character type
  - char

Value stored in memory location

Primitive Types

- Integer types
  - `byte` → 8 bits, 0, [-128, 127])
  - `short` → 16 bits, 0, [-32768, 32767])
  - `int` → 32 bits, 0, [-2147483648, 2147483647])
  - `long` → 64 bits, 0, [-9223372036854775808, 9223372036854775807]
- Floating-point types
  - `float` → 32 bits, 0.0, [+- 3.4x +38, +- 1.4x -45])
  - `Double` → 64bits, 0.0, [+- 1.7x +308, +- 4.9x -324])

Primitive Types

- Logical type
  - `boolean` → `true, false`
- Character type
  - `char` → `'\u0000'` to `'\uffff'` (or 0 – 65535)
    - Represents a unicode character:
      - 0-9
      - a-z, A-Z
      - latin symbols
      - greek symbols
      - symbols form other languages)

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`

```
String name = "John";

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, (length – 1)

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


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, (length – 1)

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


Conditional Branching

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

```
if (BooleanExpression)
   {Block1 of Statements}
if (BooleanExpression)
   Single Statement;
```


Conditional Branching

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

```
if (BooleanExpression)
   {Block1 of Statements}
if (BooleanExpression)
   Single Statement;
```


Conditional Branching – if-else ladder

```
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}
```