# Java Session 1

1/13/2025

# Biggest Differences from Python

- **Strict Typing**
  - Python type hints were a friendly suggestion/reminder.
  - Java requires all variables and functions to indicate a data type and will refuse to compile or run if the data types aren't followed exactly
- **Punctuation**
  - Number of tabs no longer have any meaning. All that matters now is the brackets `{}`
  - Need a semicolon `;` at the end of every line
- **~~Python File~~ → Java Class**
  - With Python you can have entire files or entire applications that have no classes
  - With Java *every* file is a class
- **Python function = Java class method**
  - With Python you can have functions that exist outside of a class
  - With Java *every* "function" must be created inside of a class

# Modifiers

**Used whenever you:**

- declare a new class field or variable
- define a method
- create a class

**Access Modifiers**

- optional for variables, methods and classes. But it's best practice to use them
- controls which files in your project can see or use the given variable, field, method or class

**Data Type Modifiers**

- always required for variables and methods. not used for classes.
- indicate the data type a variable will hold or the data type a method will return

**Other Modifiers**

- `final` **-** makes the variable, method or class immutable
- `static` - means the variable or method will be shared across *all* instances of the class. (can also be used for classes in certain circumstances)

# Access Modifiers

- when in doubt start `private` then work your way up to `public` if needed
- class fields should always be private, if you want to allow read/write operations then create public methods.
- methods that don't need to be used elsewhere should be private.

| | |
|---|---|
| `private` | only accessible inside of the current java class |
| | if you don't put an access modifier the default is "package private" |
| `protected` | only accessible by members of the same package (folder) |
| `public` | accessible to everybody |

# Data Type Modifiers

- these are just a few of the most common data type modifiers

| | |
|---|---|
| `void` | only for methods, indicates that it will return nothing |
| `int` | whole numbers |
| `float` | decimal numbers |
| `boolean` | true or false |
| `String` | a sequence of characters |

# Local Variables

- Created inside of a method so it has local scope
- First you need to <u>declare</u> the variable with a data type and a unique name
- Then you need to <u>initialize</u> the variable by assigning a value to it for the first time
  - This can be done in two separate lines:

```
int myNumber;     //declare the variable
myNumber = 24;        //initialize the variable
```

  - Or on the same line:

```
float pie = 3.1415;   //declare and initialize the variable
```

# Classes

- **Field**
  - variables that holds data about the class
  - sometimes referred to as attribute or property
- **Constructor(s)**
  - just like `__init__` in python
  - instructions for how to create a new instance of the class
- **Methods**
  - aka functions

```java
public class Pet{

    // class attributes
    private String name;
    private String species;
    private int age;

    // constructor
    public Pet(String petName, String petSpecies){
        this.name = petName;
        this.species = petSpecies;
        this.age = 1;
    }


    // method to print info
    public void printPet(){
        System.out.println("My pet " +
                           this.name +
                           " is a " +
                           this.species);
    }

}
```

# Class Field

- You always need to <u>declare</u> the class field with an access modifier, data type and a unique name at the very beginning of the class

```
private String name;
private int age;
```

- You can then <u>initialize</u> class fields inside of the constructor

```
public Pet(String petName){
    this.name = petName; //use param to initialize the name field
    this.age = 1 //use default value to initialize the age field
}
```

- Or you can <u>initialize</u> the field on the same line you declare it to set a default value for the field. This value can be replaced/modified later in the constructor or in one of the class methods

```
private String species = “rock”;
```

# Class Methods

**Accessor / Getter**

```
public int getAge(){
    return this.age;
}
```

**Mutator / Setter**

```
public int setAge(int newAge){
    this.age = newAge;
}
```

**Other Methods**

- methods that calculate or transform a value and then returns it
- method that doesn’t change the object but has side effects (such as printing)
- private helper methods for cleaner code

# Method Overloading

**ALLOWED** - different number of parameters for each method

```
public int doMath(int x){
    return x * x;
}
```

```
public int doMath(int x, int y){
    return x * y;
}
```

**ALLOWED** - same number of parameters but different data types

```
public int doMath(int x){
    return x * x;
}
```

```
public int doMath(String num){
    int x = Integer.parseInt(num);
    return x * x;
}
```

**NOT ALLOWED** - same number of parameters and of the same datatype

```
public int doMath(int x){
    return x * x;
}
```

```
public int doMath(int z){
    return z + z;
}
```

# Constructor Chaining

- similar to method overloading - going to create multiple definitions for the same thing, it will behave similar to optional parameters in Python

```
public Pet(String petName){
    this(petName, 1);
}
```

```
public Pet(String petName, int petAge){
    this.name = petName;
    this.age = petAge;
}
```