Object-Orientated Programming

CMM24

RGU (2023)

Session 7 (Object Orientated Programming)

Introduction to Classes:

Design

Implementation

CMM024

Summary

- **Discuss what is a class**
- **Discuss the different elements of a class**
- **Go through an example (Modelling a Person)**
- **Testing a class**

What we learnt so Far

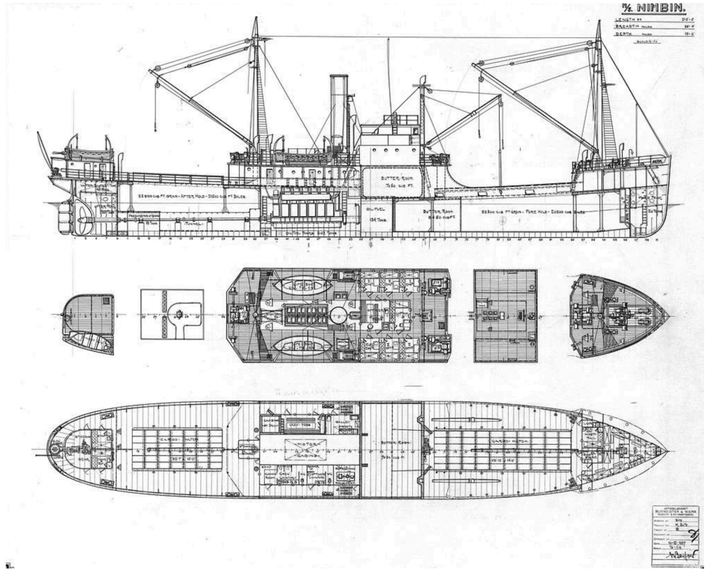
- We dealt with procedural programming the past sessions
  - Contains data and operations, both of which are separated
  - The key concept is the Procedure, which is a set of operations which accomplish a set of operations (procedures) that act on the data by passing the data between procedures to perform what the task that is required
- This can lead to
  - An increase in complexity if the programs contains many procedures
  - Also problems with data security

New Programming Paradigm

- The new paradigm we will discuss is called Object orientated programming
- The main key and important concept is the Object and object creation
- Object Orientated Programming difference with procedural programming is that this is a software entity that contains both data and the operations that act on the data
    - The data i.e. object instances and attributes are called Fields and the operations that acts on that data (i.e. fields) are called Methods
    - Both Fields and Methods belong to the **Object**! This is what we call encapsulation

What are classes?

- In Java, objects are produced from pre-programmed design "*templates*" called Classes.
- A class is a blueprint which gives complete instructions for creating a specific kind of object that will have its own data as well as its own methods that will perform tasks on the object data (change the data when the object is created) We refer to this as the object's functionality.
- To create an object, you would need to know:
  - The data the object needs
  - The methods it possess to perform some tasks on the data
  - The class also needs to know how to create one of the objects (i.e. an instance of the class).



How do we name classes?

- In Java, there is a convention when we name a class i.e. using the camelCase convention (always start with an uppercase letter)
    - ✦ Person, Student, PostGradStudent, UnderGradStudent, Server, Network
    - ✦ You often see an underscores used in the naming PostGrad_Student, UnderGrad_Student
- The java file MUST have the same name as the name of the Class
    - ✦ PostGrad_Student.java for the class PostGrad_Student
- By default, a class should be public

Data and Methods visibility

- Another important concept is the class is access visibility
    - ✦ Need to decide which fields and methods can be accessed externally or kept out I.e. hidden.
- The way we do this is to use an access (or visibility) modifier. The two modifiers we will use most commonly are:
    - ✦ private: visible only within the class (i.e. to internal code)
    - ✦ public: visible to both internal and external code
- In terms of programming you will often have to decide what should be hidden (private) and what should be visible (public) both for the object's attributes and the methods

A class is structured

- As mentioned earlier you must consider a class as a blueprint, which when used, will allow you to create a specific object that we call object instance
  - ✳ It must be structured!

Class visibility
It is public so external code code can access it, which is needed to create an object instance instance

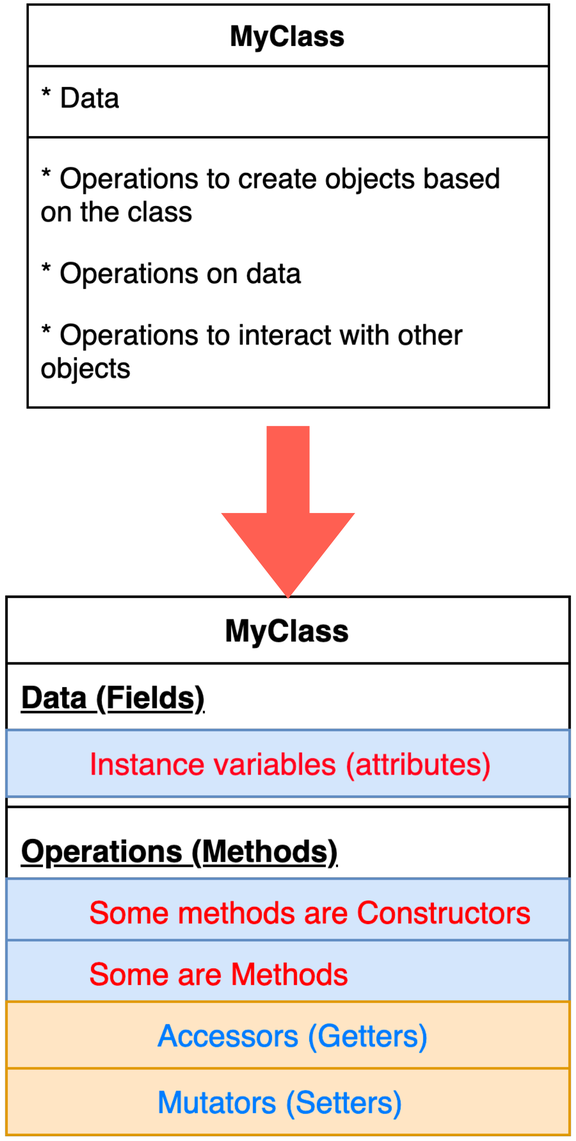
```
/** A class to represent a person
 *  @author  Jean-Claude Golovine
 *  @version 1.0: 21 July 2023
 */


public class Person {

    //the code for this person class
    //will be entered here between the
    //curly brackets

}
```

Class Header

Class name: Person
Filename: Person.java

Class structure

- A class has to contain all the information necessary to create an object both in terms of data and functionality. This means:
  - It has to specify what data the object should hold
  - It needs to specify what operations are permissible on that data
  - It needs to know how to create an instance of the class
- This gives a standard structure to a class
- To model a class we use Unified Modelling Language (UML)
  - A box with 3 compartments



Class structure

- In a simple way, a person has:
  - Full name
  - Age
  - Address
- You need to think about what Data Type that relate best to the fields!

Person
What defines a person?

- Fields, instance attributes (there are variables etc)
- A person has a name which can be held in a string
- A person usually has an address which can be held also into a string
- A person has an age which usually is held in a integer (a whole number)
- This is what we call the data for that Person object

UML

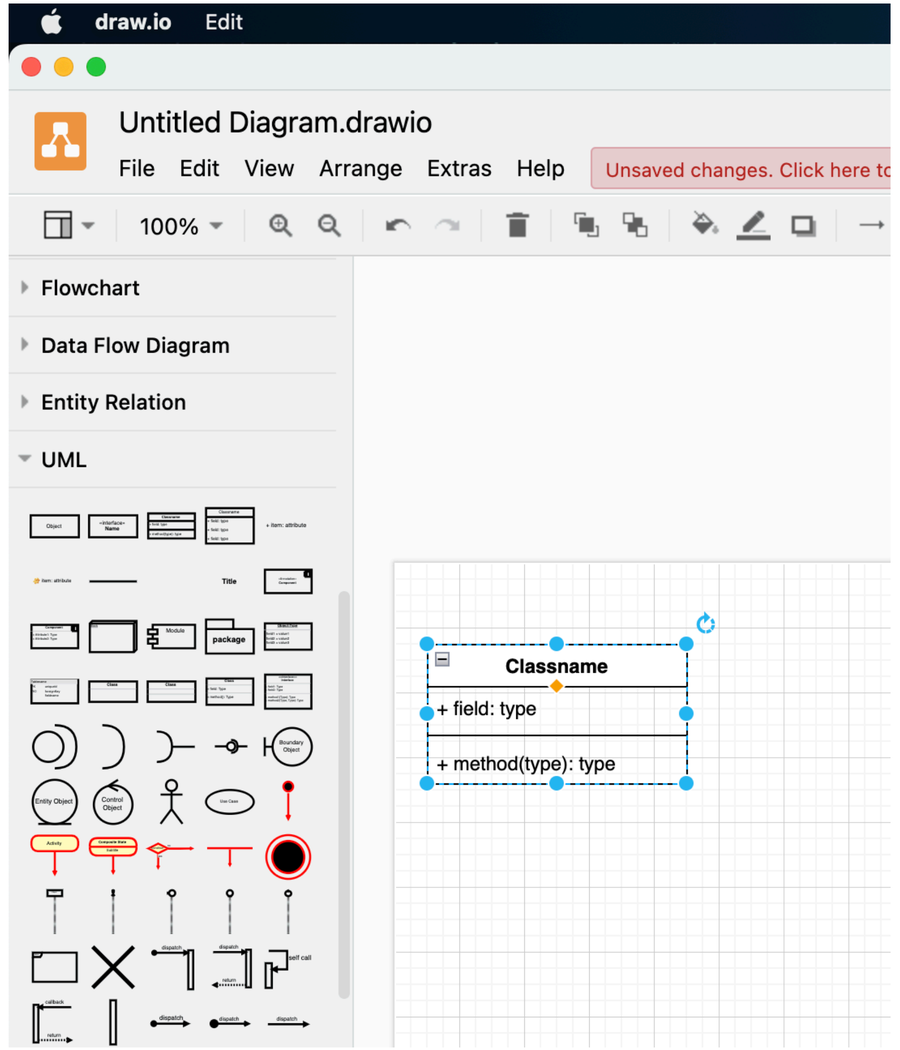
- The Unified Modelling Language is often used to model a class
- This is what we will use from now on before coding the class!
- A class in UML has 3 compartments
  - One for the class Name
  - One for the Data (object attributes & fields)
  - One for Functionality (methods and constructors)

Person
UML Model

+Person
- name: String - currentAddress: String - age: int
Note: In UML the name of the attribute comes before the datatype. When coding it is the opposite

UML

- There is a free online software that is very useful for modelling using UML.
- It is called DrawIO
- Very useful and all the UML diagrams you see in these slides were created using DrawIO



Person

How to create a person object?

- To create an object from a class you need to invoke a constructor for that class
  - ✦ It is up to you to program it.
  - ✦ You must ask yourself what information do I have to provide to initialise the fields that define that person I.e.
    - Full name, Age, Address
  - ✦ This information will come as parameters inside the constructor brackets
  - ✦ If you don’t pass this information, you person will be nameless, ageless and will have no address

```
public Person(String name, String address, int age){
     ……
}
```

Person

How to initialise the person fields?

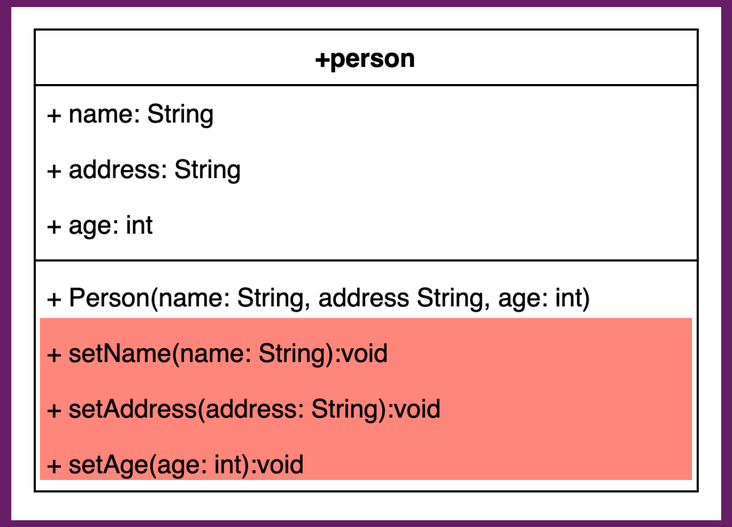
- To initialise the person fields you must use the parameters you have coded in the constructor
  - ✦ We use this to refer to the object's data fields
  - ✦ You do not have to use the same name for a parameter as the name of the class field

+person
+ name: String + address: String + age: int
+ Person(name: String, address String, age: int)

```
public class Person {

    private String name, currentAddress;
    private int age;

    public Person(String name, String address, int age){
        //to avoid ambiguities, we use this to tell Java it is
        //the instance variable rather then the parameter that
        //is set
        this.name = name;
        this.currentAddress = address;
        this.age = age;
    }
}
```

```
public class Person {

    private String name, currentAddress;
    private int age;

    public Person(String myName, String address,
        //to avoid ambiguities, we use this to t
        //the instance variable rather then the
        //is set
        this.name = myName;
        this.currentAddress = address;
        this.age = age;
    }
}
```

- Once the constructor(s) is coded, the next tasks is to create methods that will allow to change the value stored in the object fields
  - ✦ They must have a parameter that will hold the new value
  - ✦ The value of the parameter is used to set the new value of the object's field
  - ✦ We still use this to ensure the value set is the field

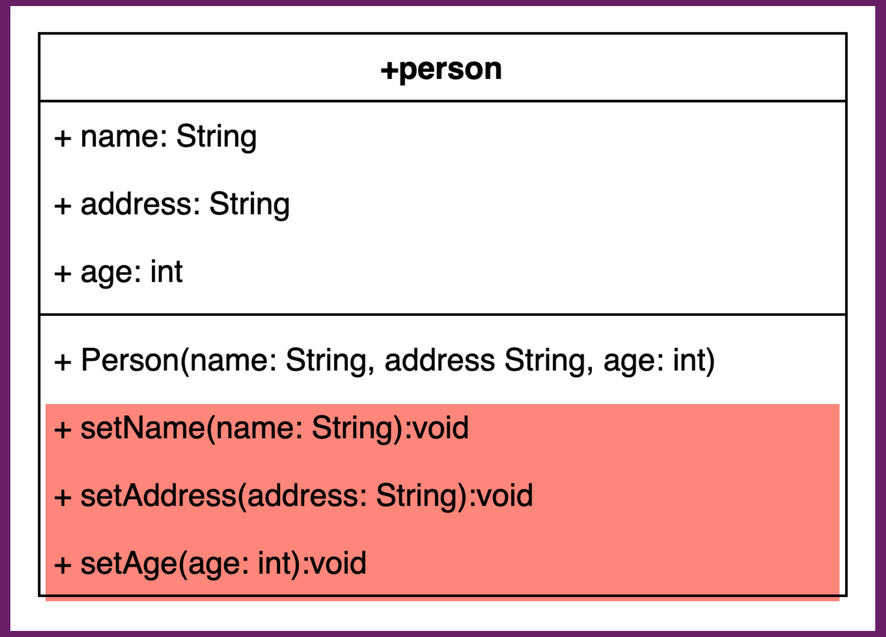


Person

How to set the person fields?

```
public class Person {

    private String name, currentAddress;
    private int age;

    public Person(String name, String address, int age){
        this.name = name;
        this.currentAddress = address;
        this.age = age;
    }
    public void setName(String name){
        ……
    }

    public void setAddress(String address){
         ……
    }

    public void setAge(int age){
         ……
    }
}
```

- We can now use the same method as we did in the constructor to set the values using the parameters. We use this
- Remember that these methods return nothing so the return type is void



Person

How to set the person fields?
By using setter methods

```
public class Person {

    private String name, currentAddress;
    private int age;

    public Person(String name, String address, int age){
        this.name = name;
        this.currentAddress = address;
        this.age = age;
    }
    public void setName(String name){
        this.name = name;
    }

    public void setAddress(String address){
        this.currentAddress = address;
    }

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

- Now that the setter methods are coded, we can program the getter methods to return the values stored in class fields
- These methods will return a value of the same data type as the field. Hence no void but in our example String or int. We MUST use the return keyword
- We don't need parameters!

+person
+ name: String + address: String + age: int
+ Person(name: String, address String, age: int) + setName(name: String):void + setAddress(address: String):void + setAge(age: int):void + getName():String + getAddress():String + getAge():int

Person

How do you retrieve a person fields? Using getter methods

```
public class Person {

    private String name, currentAddress;
    private int age;

    public Person(String name, String address, int age){
        this.name = name;
        this.currentAddress = address;
        this.age = age;
    }

   ///.....

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

   public String getAddress(){
      return this.currentAddress;
   }

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

- There are other types of methods one can create to change the data of an object. There are called transformer methods.
- Some may have parameters whilst others may not have any. Often they will not return any values and thus their return datatype will be void

+person
+ name: String + address: String + age: int
+ Person(name: String, address String, age: int) + setName(name: String):void + setAddress(address: String):void + setAge(age: int):void + getName():String + getAddress():String + getAge():int + decreaseAge():void + increaseAge():void

Person

How to transform a person field? Using transformer methods

```
public class Person {

    private String name, currentAddress;
    private int age;

    public Person(String name, String address, int age){
        this.name = name;
        this.currentAddress = address;
        this.age = age;
    }

   ///.....

    public void increaseAge(){
       this.age++;
    }

    public void decreaseAge(){
       this.age–;
    }
}
```

- There is a special method that is very useful which can be used for all class instances (i.e. object). It is called toString(). It return a String
- It is often used to return the state of object fields I.e. value stored in these fields.
- You can use this methods to return specific information about the object

+person
+ name: String + address: String + age: int
+ Person(name: String, address String, age: int) + setName(name: String):void + setAddress(address: String):void + setAge(age: int):void + getName():String + getAddress():String + getAge():int + decreaseAge():void + increaseAge():void + toString():String

Person

How to return a formatted string with information about the person fields?

```
public class Person {

    private String name, currentAddress;
    private int age;

    public Person(String name, String address, int age){
        this.name = name;
        this.currentAddress = address;
        this.age = age;
    }

   ///.....

     public String toString(){
           return "Name: " + this.name +
              " Current Address: " + this.currentAddress +
              " Current Age: " + this.age;
    }
}
```

Person Class

Object's attributes & constructor

```
public class Person {

    private String name, currentAddress;
    private int age;

    public Person(String name, String address, int age) {
        this.name = name;
        this.currentAddress = address;
        this.age = age;
    }
```

Setter methods

```
    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(String address) {
        this.currentAddress = address;
    }

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

Getter methods

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

    public String getAddress() {
        return this.currentAddress;
    }

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

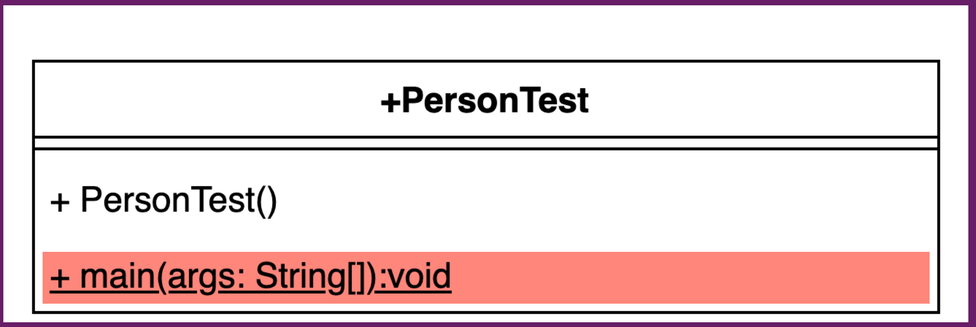
Transformer methods

```
    public void increaseAge() {
        this.age++;
    }

    public void decreaseAge(){
       this.age--;
    }
```

toString() method

```
    public String toString(){
           return "Name: " + this.name +
            "Current Address: " + this.currentAddress +
                " Current Age: " + this.age;
    }

}
```

- Creating a class and subsequently creating objects derived from that class is one thing. But how do we create an object and test our class?
- We can create a new class to test out Person class for instance!
- All the methods we have created in the Person class belong to the individual instances of the class I.e. the objects created.
- The main method does not belong to the objects that are created but it belongs to the class.
- This is why we used it because it makes the class runnable!
- In UML. We note that a method is static by underlining the line



Person

How to test a class?
By constructing a object from that class and invoke it functionality

Testing your classes

- We have created many classes in previous sessions and use the main(String args) method to run our code but we simply use the main method only.
- In this session we created class modelling a person.
- How do we test this class
    - ✦ We create a Test class and use its main method to create object instances of Person!

Person
What defines a person?

Person class

PersonTest class

```
Public static void main(String[] args) {
    Person personObj = new Person
    Display personObj information
    Increase age
    Redisplay  personObj information
}
```

Test classes / using the main method

- The main method makes your class runnable. We often use Test classes and its main method to create objects I.e. main instances.

```
public class PersonTest {
    public static void main(String[] args) {
        // Create a person object
        Person personObj = new Person("Claude", "Aberdeen", 21);
        // Display details of the object using the toString method
        System.out.println(personObj.toString());
        // Increase the age
        personObj.increaseAge();
        // Re-Display details of the object using the toString method
        System.out.println(personObj.toString());
    }

}
```

```
/JavaVirtualMachines/liberica-jdk-17-full.jdk/Contents/H
laude/CMM024Lectures/CMM024_Lectures/bin PersonTest
Name: Claude Current Address: Aberdeen Current Age: 21
Name: Claude Current Address: Aberdeen Current Age: 22
jean-claude@MacBook-Pro-2 CMM024_Lectures %
```

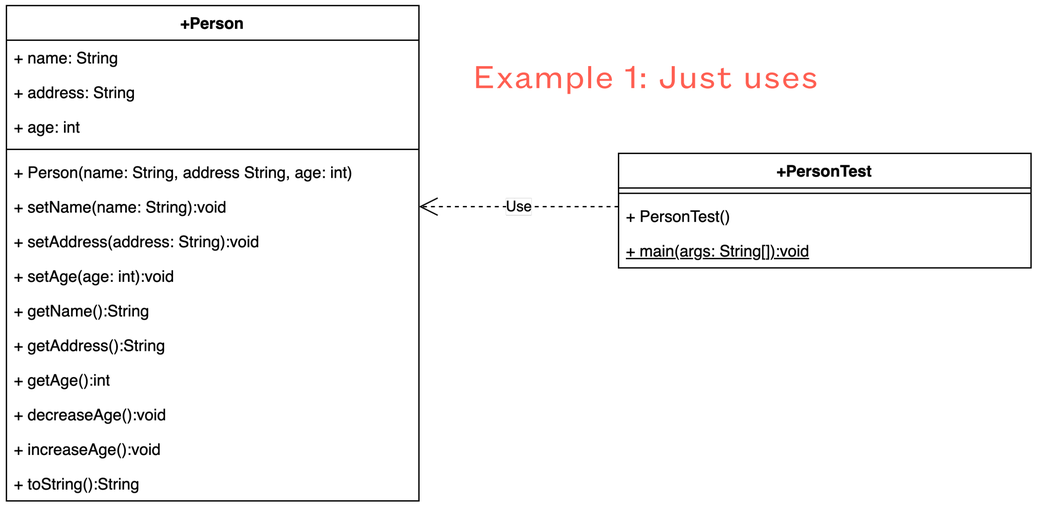
Test classes / ArrayList & for loop

```
import java.util.ArrayList;
public class PersonTest {
    public static void main(String[] args) {
        //create an ArrayList to hold the member as a group
        ArrayList<Person> group = new ArrayList<>();
        // Create 3 people for that group
        Person person1 = new Person("Claude", "Aberdeen",18);
        Person person2 = new Person("Roger", "Aberdeen", 21);
        Person person3 = new Person("Shahana", "Aberdeen", 23);
        //Add each person to the group i.e. ArrayList
        group.add(person1);
        group.add(person2);
        group.add(person3);
        //display a blank line
        System.out.println();
        //display the details for each person by invoking toString
        for (Person member: group){
             System.out.println(member.toString());
        }
        //display a blank line
        System.out.println();
        //invoke some functionality to change the data of some person
        person2.increaseAge();
        person1.setAge(63);
        //re-display the details for each person by invoking toString
        for (Person member: group){
             System.out.println(member.toString());
        }
         System.out.println();
    }
}
```

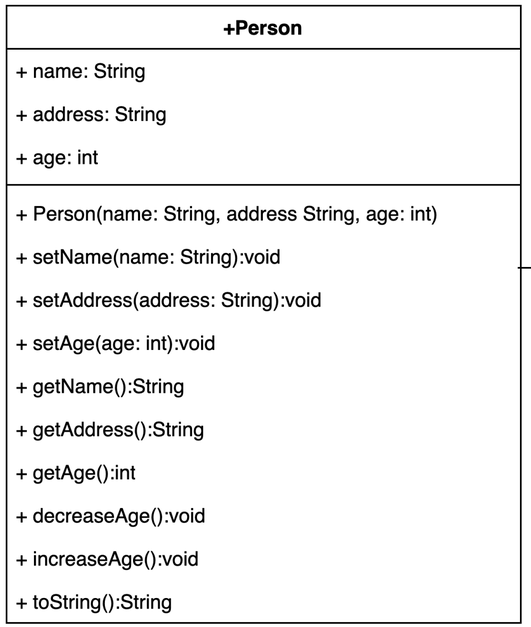
- We can use an ArrayList to store multiple Person objects
- We can use a for loop to iterate that list and display the object details by invoking toString() for each person

```
Name: ClaudeCurrent Address: Aberdeen Current Age: 18
Name: RogerCurrent Address: Aberdeen Current Age: 21
Name: ShahanaCurrent Address: Aberdeen Current Age: 23

Name: ClaudeCurrent Address: Aberdeen Current Age: 63
Name: RogerCurrent Address: Aberdeen Current Age: 22
Name: ShahanaCurrent Address: Aberdeen Current Age: 23
```

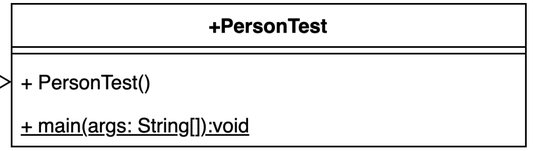
Test classes / using the main method



Example 1: Just uses



Example 2: Weak Aggregation



Next week

- **Introduction to Inheritance**