CMSC203 Object-Oriented Programming - Guide

**Task #1** – Creating a basic *Person* **class**

1. Create a class *Person* in Java that does not possess a main method
2. Modify *Person* to have the following attributes:

* A string variable called name
* An integer variable called age

A screenshot of a cell phone

Description automatically generated

**Task #2** – Create instances of *Person*

1. Create a class *Person*Demo in Java that possesses a main method

A picture containing screenshot

Description automatically generated

1. In order to use *Person* class you need to define variable of type Person

Person **a** ;

1. Now use the new keyword to create an **object** of the class *Person*, and assign the object to the variable from Step 2 so we can invoke the object in the future

**a** = new Person ( ) ;

[This is called instantiation of an object]

1. The variable **a** now refers to an object of the class *Person* and we will thus refer to it as such
2. Being an object of class *Person*, **a** now possesses its own individual fields name and age as outlined in the *Person* class definition from which **a** was created
3. Assign a String “Todd” representing a name to the name field and an integer literal “18” representing an age to the age field belonging to the new *Person* object **a**

(Hint: try typing “a.” and see what Eclipse suggests)

**Task #3** – Use access modifiers

1. Go back in your file to the declaration of variables name and age. These attributes (fields/variables) are by default set to **public**, meaning any method or file can change or access them
2. Append the keyword **public** to the beginning of each variable declaration

**public** String name ;

**public** int age ;

1. This keyword is an **access modifier**. It specifies what can and cannot change and/or access content of an attribute.

Ex: In *Person*Demo, try to access the name field of *Person* object **a** and print it

( Hint: System.out.println(a.name); )

Notice that this should work and whatever name you assigned to the name field (in this case Todd) should be printed on the console output screen

1. Now replace the keyword public in the declarations with the keyword **private**

**private** String name ;

**private** int age ;

1. By specifying these variables as **private** members of the class *Person*, we ensure that nothing (outside of the *Person* object to which these private members belong) can change or access these members’ contents

Ex: If you save what we have done so far with regard to files *Person* and *Person*Demo, the *Person* object **a** will now be inaccessible (not visible) to other classes (like *Person*Demo)

Run *Person*Demo again, and the name field should now be inaccessible

**Task #4** – Getter and Setter Methods

1. Now that the fields (variables) of *Person* object **a** are private, we need a way to access them in external classes like *Person*Demo
2. Being specified as a private field of an object means nothing can change or access it except for a method called by the very same object to which the field belongs
3. In the *Person* class, define a method that when invoked will change (or set) the String field name to possess a new String

This is called a **setter** (or mutator) **method**

It is often named “set\_\_\_\_” with the blank filed in by the name of the field to be set/changed

1. Repeat Step 3 for the integer field age

A screen shot of a smart phone

Description automatically generated

1. Go back into the *Person*Demo class and where you originally set the name field of *Person* object **a** to “Todd”, do the same but this time using the setter method

Remember that in order to call an object’s method you must first invoke the name of the object

1. Repeat Step 5 for the integer field age
2. We can now successfully change the contents of a private field from outside the object to which it belongs, but we still cannot retrieve it (as indicated by the error displayed in *Person*Demo where we sought to print a.name in Task #3). We must now create another *Person* class method, but this time with the purpose of retrieving/returning the contents of a private field.
3. In the *Person* class, define a method that when invoked, will return the contents of the String field name

This is called a **getter** (or accessor) **method**

1. Repeat Step 8 for the int field age

A screenshot of a cell phone

Description automatically generated

1. Go back into *Person*Demo and where we initially sought to print the contents of the name field, instead invoke and pass the getter method for the name field.
2. There should now be no error in Eclipse

**Task #5** – Constructors

1. Notice that in *Person*Demo when we instantiated the *Person* object **a**, we used the new keyword in addition to what appeared to be a method Person()
2. In the world of Object-Oriented Programming, every class (whether programmer-defined or included in the Java library) possesses a special method called a **constructor** that, when invoked, creates a new object of the class in memory
3. Every class possess a default constructor (whether visible to the programmer or not)
4. Every constructor is a public method that possess a name identical to that of its class
5. Return to the *Person* class and define a no-argument constructor that sets the name field to an empty string “” and the age field to 0A screen shot of a smart phone

   Description automatically generated
6. Like any other method, constructors may possess parameters that, when passed to the constructor, set the fields of a newly instantiated object to some desired initial values

The only difference between a no-arg constructor and a parameterized constructor is the inclusion (or lack thereof) of parameters, thus the constructors can be considered overloaded methods

1. Create a parameterized constructor that takes a String and an integer value and pass them to the name and age field respectivelyA screen shot of a smart phone

   Description automatically generated