Java Objects

Task1

Let us create a class named Person. The class will exist in its own Person.java file.

```
public class Person{

    public static void main(String[] args){

    }
}
```

Task2: Attributes

The class must have some attributes and methods depending on the requirements of a program. Let us decide that each person has a name and an age. It feels natural to represent the name as a String and the age as an integer. Let us add this to our schematics:

```
public class Person{
    private String name;
    private int age;

    public static void main(String[] args){

    }
}
```

Above, we defined that all objects created from the Person class have a name and an age. Defining attributes is done in a quite similar fashion as with normal variables. In this case though, there is the keyword *private* in front. This keyword means that name and age will not show outside of the object, but are instead hidden within it. Hiding things within an object is called *encapsulation*.

Variables defined within a class are called object variables, object fields or object attributes. A beloved child has many names.

So, we have defined the schematics – the class – for the person object. All person objects have the variables name and age. The 'state' of the objects is determined by the values that have been set to its variables

Task3: Constructor Methods

Constructors without parameter

The constructor always has the same name as the class. Let's begin by defining a simple constructor that simply sets the name and age of each person to the same values. This version is shown here:

```
public class Person{
    //...

    public Person() {
        this.age = 0;
        this.name = "A";
    }

    public static void main(String[] args){
        Person bob = new Person();
    }
}
```

In the code above, the class is Person and the constructor is public Person().

Constructors with parameter

While the **Person()** constructor in the preceding example does initialize a Person object, it is not very useful—all persons have the same age and name. What is needed is a way to construct Person objects of various names and ages. The easy solution is to add parameters to the constructor. As you can probably guess, this makes it much more useful. For example, the following version of Person defines a parameterized constructor that sets the name and age of a person as specified by those parameters. Pay special attention to how Person objects (e.g. bob) are created.

```
public class Person{
    //...

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

    public static void main(String[] args){
        Person bob = new Person("Bob", 10);
    }
}
```

A few notes: within the constructor there is a command this.age = age. Through it, we set a value for this particular object; we define the internal variable age of "this" object. Another command we use is this.name = name;. Again, we give the internal variable called name the String that is defined in the constructor.

The variables age and name are automatically visible in the constructor and elsewhere in the object. They are referred to with the this prefix. Due to the private modifier, the variables cannot be seen from outside the object.

One more thing: if the programmer does not create a constructor for her class, Java will automatically create a default constructor for it. A default constructor is a constructor that does nothing. So, if you for some reason do not need a constructor you do not need to write one.

Task4: Getter and Setter Methods

In Java, getter and setter are two conventional methods that are used for retrieving and updating value of a variable. The naming scheme of setter and getter should follow *Java bean naming convention* as follows: **getXXX()** and **setXXX()**, where *XXX* is name of the variable. Generally, we write one setter and one getter for a single variable.

As the Person class has 2 variables, we have defined two sets of setter and getter methods: one set for name variable and another for age variable.

```
public class Person{
    //...

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

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


    public static voidmain(String[] args){
        Person bob = new Person("Bob", 10);
        System.out.println(bob.getAge());
        bob.setAge(50);
        System.out.println(bob.getAge());

    }
}
```

Task5: More Methods(Printing variables inside a single method)

We already know how to create and initialize objects. Also we have seen how to write setter and getter methods for a class. However, objects are useless if they cannot do anything other than updating their respective variables. Therefore, objects should have methods. Let us add to the Person class a method that prints the object on the screen:

```
public class Person{
    //...

    public void printPerson() {
        System.out.println(this.name + ", age " + this.age + " years");
    }

    public static void main(String[] args){
        Person bob = new Person("Bob", 10);
        System.out.println(bob.getAge());
        bob.setAge(50);
        System.out.println(bob.getAge());

        bob.printPerson();
    }

}
```

Task6: toString() Method

We have been guilty of bad programming style; we have created a method that prints an object, *printPerson*. The recommended way of doing this is by defining a method that returns a "character string representation" of the object. In Java, a method returning a String representation is called **toString**. Let us define this method for person:

```
public class Person {
    //...

    public String toString() {
        return this.name + ", age " + this.age + " years";
    }

    public static void main(String[] args) {
        Person bob = new Person("Bob", 10);
        System.out.println(bob.getAge());
        bob.setAge(50);
        System.out.println(bob.getAge());
```

```
        System.out.println( bob );
                    // same as System.out.println(andy.toString());

    }
}
```

The principle is that the System.out.println method requests the string representation of an object and then prints it. The returned string representation of the toString method does not have to be written, as Java adds it automatically. When the programmer writes:

System.out.println(andy);

Java completes the call during runtime to the format:

System.out.println(andy.toString());

What happens is that the object is asked for its string representation. The string representation the object is returned and is printed normally with the System.out.println command.

Now delete the previously written method *printPerson()*

Task 7: More Methods(Increasing age)

Let us create a method that can be used to increase the age of a person by one:

```
public class Person{
    //...

    public void becomeOlder() {
        this.age++;;     // same as this.age = this.age + 1;
    }

    public static void main(String[] args){
        Person bob = new Person("Bob", 10);
        System.out.println(bob.getAge());
        bob.setAge(50);
        System.out.println(bob.getAge());

        System.out.println( bob );

        Person brian = new Person("Brian", 20);
        Person martin = new Person("Martin", 30);


        bob.becomeOlder();
        brian.becomeOlder();

        System.out.println(bob.getAge());
```

```
        System.out.println(brian.getAge());

    }
}
```

Task8: More Methods(Checking adult or not)

Let us create a method for person that can figure out if a person is an adult. The method returns a boolean – either true or false:

```
public class Person{

   //...

    public boolean isAdult(){
        if ( this.age < 18 ) {
            return false;
        }

        return true;
    }

    public static void main(String[] args){
        Person bob = new Person("Bob", 10);
        System.out.println(bob.getAge());
        bob.setAge(50);
        System.out.println(bob.getAge());

        System.out.println( bob );

        Person brian = new Person("Brian", 20);
        Person martin = new Person("Martin", 30);

        bob.becomeOlder();
        brian.becomeOlder();

        System.out.println(bob.getAge());
        System.out.println(brian.getAge());

        int i = 0;
        while (i <= 30){
            brian.becomeOlder();
            i++;
        }
        System.out.println(brian.getAge());
    }
```

}

Task8: More Methods(Finding Body Mass Index)

Let us continue with the class Person. We would be interested in knowing the body mass index of a person. To calculate the index, we need to know the height and weight of the person. We add for both height and weight object variables and methods that can be used to assign the variables a value. When this is in place, we add a method that calculates the body mass index.

Here is the class Person after the changes (only the parts affected by the change are shown):

```
public class Person {
   //...

    private int weight;
    private int height;

    public Person(String name, int age) {
        this.age = age;
        this.name = name;
        this.height = 0;
        this.weight = 0;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public void setWeight(int weight) {
        this.weight = weight;
    }

    public double bodyMassIndex(){
        double heightDividedByHundred = this.height / 100.0;
        return this.weight / ( heightDividedByHundred
                              * heightDividedByHundred );
    }

    public static void main(String[] args) {
        Person bob = new Person("Bob", 10);
        System.out.println(bob.getAge());
        bob.setAge(50);
        System.out.println(bob.getAge());

        System.out.println( bob );
```

```
        Person brian = new Person("Brian", 20);
        Person martin = new Person("Martin", 30);

        bob.becomeOlder();
        brian.becomeOlder();

        System.out.println(bob.getAge());
        System.out.println(brian.getAge());

        int i = 0;
        while (i <= 30){
            brian.becomeOlder();
            i++;
        }
        System.out.println(brian.getAge());

        Person matti = new Person("Matti", 10);
        Person john = new Person("John", 20);

        matti.setHeight(180);
        matti.setWeight(86);

        john.setHeight(175);
        john.setWeight(64);

        System.out.println(matti.getName() +",bodymassindex:"+
                           matti.bodyMassIndex());
        System.out.println(john.getName() + ",bodymassindex:"+
                           john.bodyMassIndex());
    }
}
```

Task 9: Calling methods inside a method

Objects can also call its own methods. Let us assume we would like to include bodyMassIndex variable in the string representation of the person objects. Instead of calculating the body mass index in the toString method, a better idea is to call the method bodyMassIndex from the toString method:

```
public String toString() {
    return this.name + ", age " + this.age + " years, my" +
           "body mass index is " + this.bodyMassIndex();
}
```

As can be seen, an object can call its own method by prefixing the method name with this and dot. The this is not necessary, so also the following works:

```
public Sring toString() {
    return this.name + ", age " + this.age + " years, my" +
           "body mass index is "+ bodyMassIndex();
}
```

Now it is time to continue practising programming.