

1.1 OBJECTS AND CLASSES

All **objects** are unique. But at the same time they are also part of a class of objects (for example, a student is unique but he/she is also part of the class that is called “human beings”). The class of human beings has a set of characteristics and behaviours in common which are different from other classes such as animal or vehicle classes.

When we want to define an object in our program, we need to define its class first. An object is created from a class. If there is no class, then no object can be created.

If we take the example of student class, each student has a set of variables (such as matric number, name, age, address, etc) and methods or behaviours (such as register course, withdraw course, etc). Each of the student has his/her own set of variables and can call the methods independently. Any call to methods will change it's variable. For example, a repeated call to the register course method will increase the value of the number of courses registered.

Each object within a class has its own set of variables and methods. We can compare objects from the same class to individual student in a group. For example, imagine we have two objects, John and Brown that belong to the same class – Student. John has his own set of variables: name, address, age, etc which the values are different with that of Brown. John also keep some of his variables such as age and address as private data (which are not accessible by other students) but he may let Brown to access other data such as name and matric number. Also, John may allow Brown to execute his method such as `displayRegisteredSubjects()` for him. And at the same time, John may not allow Brown to execute certain methods such as `changeAddress()` for him.

Ever wonder what the difference between a class and primitive data type such as int or double?

- A primitive type is predefined by the language and is named by a reserved keyword. In Java, there are eight primitive types: byte, short, int, long, float, double, boolean, char.
- An object is a variable you name and define. Specifically, you define the object's state and behavior using properties. Objects attempt to model real world items and how they operate or interact with other objects.

A **class** is a blueprint that defines the variables (or attributes) and the methods common to all objects of a certain kind. In the real world, we often have many objects of the same kind. For example, your car is just one of many cars in the world. In object-oriented context, we say that your car object is an instance of the class of objects known as cars. Cars have some state (current gear, current speed, four wheels) and behavior (change gears, change speed) in common. However, each car's state is independent and can be different from that of other cars. When building them, manufacturers take advantage of the fact that cars share characteristics, manufacturing many cars from the same blueprint. It would be very inefficient to produce a new blueprint for every car manufactured. In object-oriented software, it's also possible to have many objects of the same kind that share characteristics: employee records, video clips, students record and so on. Like car manufacturers, you can take advantage of the fact that objects of the same kind are similar and you can create a blueprint for those objects.

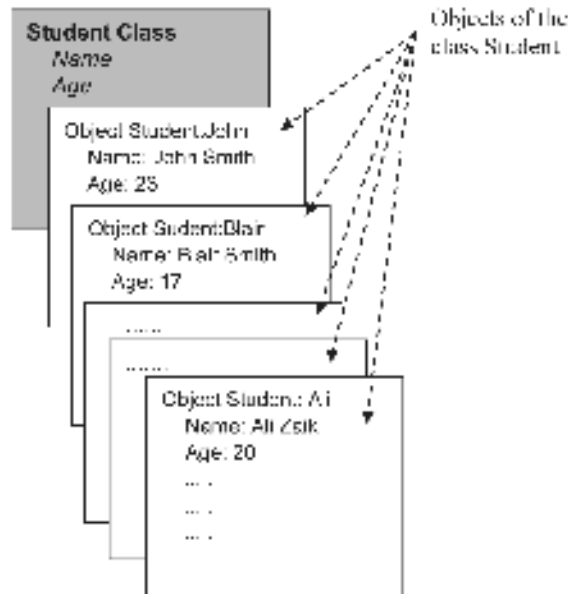


Figure 1.2: Objects created from the class Student

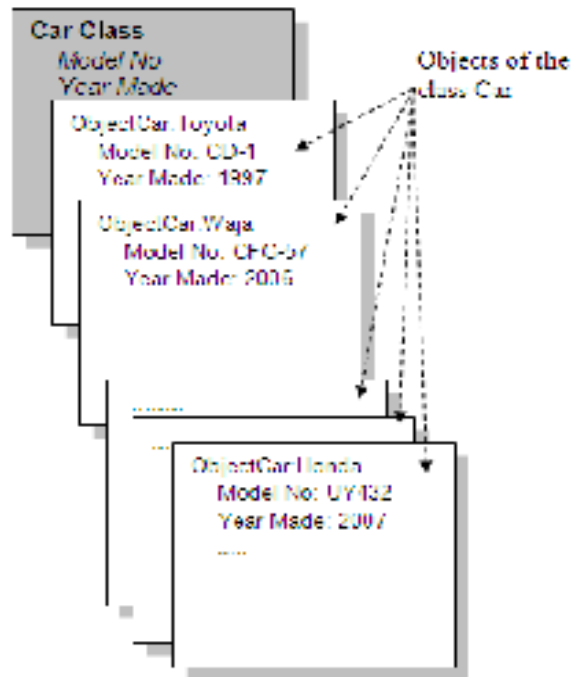


Figure 1.3: Objects created from the class Car

1.2 DEFINING CLASSES

A class is the template for an object just like data type `int`. But there is a difference between class and primitive data type such as `int` or `double`. That is when you define a class, you need to tell Java the characteristics of the class you created but when you define an `int` data type, the compiler knows what an integer is and you are not required to specify anything.

Below is the structure of a class in Java:

```
class <class_name> {  
    <attribute(s)>  
    <operation(s)>  
}
```

This declaration is divided into three components:

- **<class_name>**
This component is the class name that we declare.
- **<attribute(s)>**
This component is the list of attributes for each object created from this class. In Java, the object attributes can be defined as variables and it is called as member variables or instance variables.
- **<operation(s)>**
This component is the list of behavior for all objects created from this class. These behaviors is defined as member methods.

We define a class by using the `class` keyword followed by the class name. The class body is marked off by curly braces just like a program block. The following is Java class which does nothing.

```
public class Student {
```

The above class Student is an empty class; it does nothing. However, you can compile the class and create an object from it. To make Student a useful class, it must have some attributes (that will represent its state) and methods (that will represent its behaviour). Assume the class Student will have the following states (we just limit to EIGHT states only):

- Student's Name
- Student's matric number
- Student's year birth
- Student's age
- Fees to be paid by the student
- Number of courses registered by the student
- Name of the subjects registered by the student
- Registration status of the student

The above states are represented as attributes in a class. To declare attributes in a class use the following syntax:

```
data_type attribute_name;
```

You need to type the appropriate data type followed by the name of the attribute as shown below:

Program 1.1:

```
public class Student {  
    private String name;  
    private int matricNo;  
    private int yearBirth;  
    private int age;
```

```
private double fees;

private int numberOfCoursesRegistered;

private String[] subjectName;

private boolean registered;

}
```

Note: The keyword that is private written together with the attribute declaration in the above program is an access modifier and it's not compulsory. You will learn about access modifier in the next topic.

The above class declares eight attributes (1 String type attribute, 4 integer type attributes, 1 double type attribute, 1 String type array and 1 boolean data type).

1.3 CREATING AN OBJECT FROM A DEFINED CLASS

When you create a class you are actually describing how objects of that class look and how they will behave. You don't actually get anything from this class until you create an object of that class with the keyword new. The following are syntax to create an object from a class:

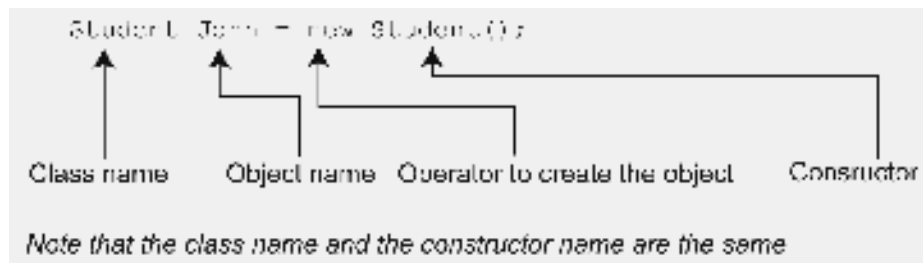
```
Class_Name Object_Name = new Constructor_Name( );
```

The following example shows how you can create an object from the Student class shown in Program 1.1 using the above syntax:

Program 1.2 (a):

```
Line 1: class obj {
Line 2:     public static void main (String[] args ){
Line 3:         Student John = new Student(); // create the object
Line 4:     }
Line 5: }
```

The meaning for all the elements stated in Line 3 in the above program is given below:



Take note that the Class_Name and the Constructor_Name are the same.

Alternatively, you also can use the following syntax in order to create an object from a class:

```
Class_Name Object_name;
```

```
Object_Name = new Constructor_Name( );
```

The following example shows how you can create an object from the Student class shown in Program 1.1 using the above syntax:

Program 1.2 (b):

```
class obj {
    public static void main (String[ ] args ){
        Student John ; //reference type
        John = new Student(); // create the object
    }
}
```

The state of the John object after execution of the Program 1.2(a) or Program 1.2(b) is diagrammatically shown below:

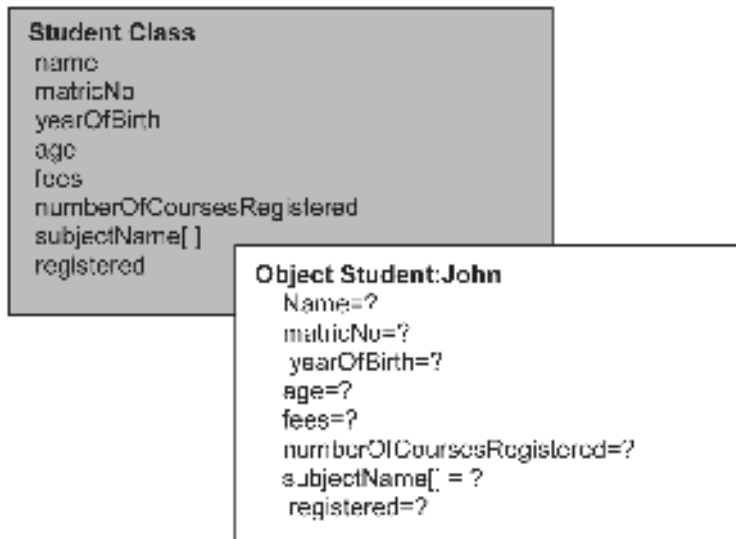


Figure 1.4: Object created from the class Student

Can you create more than one object of the same class in a program? Of course you can as long as your computer memory can support it.

1.4

CONSTRUCTORS

You could see in Figure 1.5 that the value of object attributes are not defined (i.e. the attributes are not given any value). In most cases, attributes in an object of a class need to be initialized. The best approach is by doing the initialization when creating an object. This can be done using a special method called constructor. Constructor method is a special method that can be used when an object is created using the **new** keyword. This means that an object uses the constructor method once in a lifetime. Therefore, the object initialized code is more proper if we put it in together in the constructor method.

It make sense that whenever an object is created, we have to do some initialization to set the values of instance variables for each object. In Java, the class designer can guarantee initialization of every object by providing a special method called constructor. When an object is created, its members can be initialized by a constructor method. The programmer provides the constructor and this constructor is invoked automatically each time an object of a class is created.

A constructor can do anything that a normal method can do, but usually is used merely to initialize the attributes within the object.

Since constructor is a kind of method, the format to write the constructor is almost similar to the ordinary method with some minor variations as shown below:

```
<modifier> <class_name> ([<parameter_list>]){  
    [<statements>]  
}
```

The name of the constructor method must be the same as the class name. The <parameter_list> is same as for the method declaration (refer to the Topic 3). The valid modifiers (<modifier>) for constructors are public, private and protected which will be elaborated in the next topic. The constructor method cannot have return values. The void keyword should be avoided in the header of the method even though it does not return any value.

REMEMBER:

In Java, there are two constraints when we define the constructor method in a class:

- The name of the constructor method must be the same as the class name;
- The constructor method does not have return type – not even void

Program 1.4 below is expanded and modified from Program 1.1 after a constructor has been included:

Program 1.4:

```

public class Student {
    private String name;
    private int matricNo;
    private int yearOfBirth;
    private int age;
    private double fees;
    private int numberOfCoursesRegistered;
    private String[] subjectName;
    private boolean registered= false;

    public Student (String NAME, int MATRIC, int Y_BIRTH) {
        name= NAME;
        yearOfBirth= Y_BIRTH;
        matricNo= MATRIC;
        numberOfCoursesRegistered=0;

        subjectName= new String[3]; /*create array to store the
                                     name of three subjects*/

    } //constructor
} //class

```

Notice that the constructor name and the class name are same

Constructor

You can recall that when *new* keyword is used to create a new object, the calling syntax is:

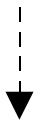
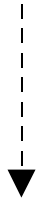
```
Class_Name Object_Name = new Constructor_Name( );
```

Notice that the constructor method is called during the creation of an object. The following

program shows how an object of the class Student has been created:

Program 1.5:

```
class obj {
    public static void main (String[] args ){
        Student John = new Student("John Smith", 2345, 1969);
    }
}
```



This statement is actually calling the constructor method available in the Student class

The effect of this new keyword:

- Allocates necessary space to store the object John which is the object of the class Student
- Invokes the constructor
- Returns reference to the appropriate object

It is not compulsory for us to assign initial values to all the attributes in the class. You could see that the attributes age and fees are not given any initial value in the constructor of the Program 1.4. Also, sometimes the initial also could be assigned during attribute declaration or in other operational methods. For example, the initial value for the attribute registered was assign during its declaration.

Besides using constructors to initialize the attributes, it also can be used to perform tasks related to creation like performing initial calculations, etc

Important Point to Note:

You may come across classes that do not have constructor method. Actually, it is not compulsory for you to include constructor method in a class. You must be careful when adopting this approach.

The compiler automatically provides a no-argument, default constructor for any class without constructor(s).

Nevertheless, it will be an error for calling a no-argument constructor when creating an object if a class already has constructor with argument unless you have already defined your own no-argument constructor in the class.

1.5 MEMBER METHODS

(You should understand Section 3.4 of Topic 3 before reading through this section. Good understanding about method elaborated in Section 3.4 will help you to understand this section better)

Do you know that merely declaring attributes and constructor in a class is not so complete?

To make your class more useful, you must include "operations" in the class. Operations in a class are represented by member methods. The method calculateAge() found in Program 1.6 is an example of member method. Remember that member method is one of the components that make up a class. What is the difference between constructor and member method? The answer is given in the following table.

Table 1.2: Differences between Constructor and Member Method

Constructor	Member Method
The name of the constructor method is same as the name of the class	The member method can be given any meaningful name except the name of the class
Constructor method cannot have any	Constructor method may or may not return

return value	value
Constructor method is used to initialize attributes	Member method is used to perform some calculations such calculate age, get maximum number, get random number, etc.

There are three types of member methods that could be written in a class namely: **query method**, **update method** and **task method**. Query method is used to access only the object's state and this operation does not change the state of a object. Example of this method are: getName(), getPay(), RetrieveID(), checkPassword()

Update method is used to update or change the value of attribute(s) of a object. This update operation will change the state of the object. Example of this method are: setName(), updateAge().

Finally, task methods is used to perform some specific task. This method may change the state of the object. Example of this method are: calculateAge(), convertToCelcius(). Unlike constructor method which is called during method creation, member method is called after an object has been created. To give you some example on member methods, we are going to expand Program 1.6 by adding the following operations:

- Calculate fees - fees is calculated based on number of subjects registered
- Capture the subject names to be registered by the students
- Display the personal information of the student

Note: Of course the list can be long but we just limit it to three operations only for the sake of the discussion.

Each of the above operation can be represented by different member methods. After taking consideration the above operations, the new revised program with member methods is shown below:

Program 1.8:

```
public class Student {
    private String name;
    private int matricNo;
    private int yearOfBirth;
    private int age;
    private double fees;
    private int numberOfCoursesRegistered;
    private String[ ] subjectName;
```

```
private boolean registered= false;

public Student (String NAME, int MATRIC, int B_YEAR) {
    name= NAME;
    yearOfBirth=B_YEAR;
    matricNo= MATRIC;
    numberOfCoursesRegistered=0;
    subjectName= new String[3];
    calculateAge();
}
private void calculateAge (){
    int currentYear=2008;
    age= currentYear-yearOfBirth;
} // calculateAge

public void calculateFee (){
    if (registered==true)
        fees=numberOfCoursesRegistered*350;
} // calculateFee

public void registerSubject (String subject1){
    if (registered==false) {
        subjectName[0]=subject1;
        numberOfCoursesRegistered=1;
        registered=true;
    }
    else
        System.out.println("You already registered");
} // registerSubject
public void registerSubject (String subject1, String subject2){
    if (registered==false) {
        subjectName[0]=subject1;
        subjectName[1]=subject2;
        numberOfCoursesRegistered=2;
        registered=true;
    }
    else
        System.out.println("You already registered");
} // registerSubject
public void registerSubject (String subject1, String subject2, String subject3){
    if (registered==false) {
        subjectName[0]=subject1;
        subjectName[1]=subject2;
```

overloaded
member
methods

```

        subjectName[3]=subject3;
        numberOfCoursesRegistered=3;
        registered=true;
    }
    else
        System.out.println("You already registered");
} // registerSubject
public void displayInfo () {
    if (registered==true){
        System.out.println("Name: " + name);
        System.out.println("Matric: " + matricNo);
        System.out.println("Year of Birth: " + yearOfBirth);
        System.out.println("age " +age);
        System.out.println("Fees: " + fees);
        System.out.println("No of courses registered:
            +numberOfCoursesRegistered);
        System.out.println("Courses registered are:");
        for (int count=0; count < numberOfCoursesRegistered; count++)
            System.out.println(subjectName[count]);
        } //if
    else
        System.out.println("Information cannot be displayed because you have not
registered");
    } //displayInfo
} //class

```

Members methods also can be overloaded and overloaded methods must have same method name but with different signatures as you could see in the methods registerSubject() of the Program 1.8 (refer to section 3.4 in topic 3 for explanation on method overloading).

1.5.1 Calling Member Methods

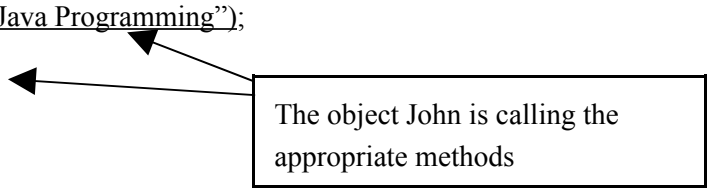
In Program 1.8, we have included few member methods that represent few operations. How this method could be called to perform their designated task? Before a member method could be called, the object of the class must be created first. Then, the appropriate method could be called using the following syntax:

```
<object_name>.member_method_name;
```

Remember that member methods are called after the object has been created. Here is how you can call member method `calculateAge()` available in the `Student` class of the Program 1.8:

Program 1.9 (a):

```
class obj {  
    public static void main (String[ ] args ){  
        Student John = new Student("John Smith", 2345, 1969);  
        John.registerSubject("Java Programming");  
        John.calculateFee();  
    }  
}
```



The object John is calling the appropriate methods

Note that the object John has not invoked all the member methods in the `Student` class. It just invoke two member methods in the `Student` class, namely `registerSubject()` and `calculateFee()`. After executing the Program 1.9(a), the object John will be in the following state:

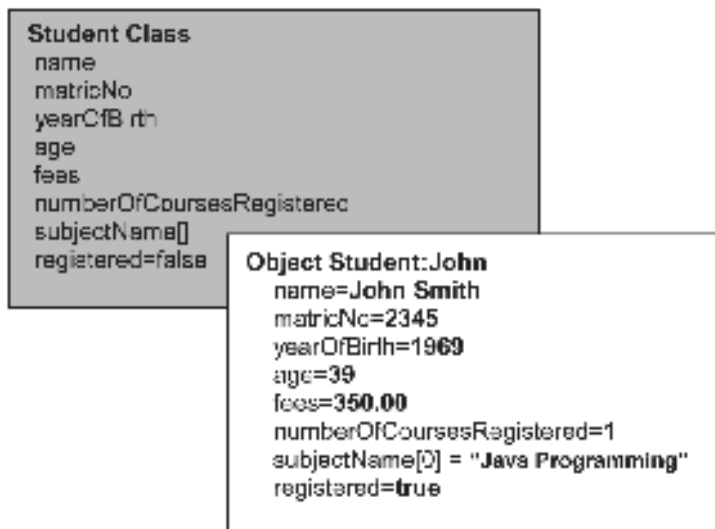


Figure 1.7: The state of object John after execution of Program 1.9(a)

Program 1.1 – Program 1.9 is not interactive (that is the inputs are not key in by the user during the program run-time). The Program 1.9(b) below shows of how Program 1.9(a) has been amended to make it interactive:

Program 1.9 (b):

```

import java.io.*;

class obj {

    public static void main (String[] args ) throws IOException {

        Pembaca read = new Pembaca();

        System.out.println("Enter name:");

        String name= read.bacaString ();

        System.out.println("Enter matric number:");

        int matric= read.bacaInt();

        System.out.println("Enter year birth:");

        int year= read.bacaInt();

        System.out.println("Enter subject name:");

        String subject= read.bacaString ();

        Student John = new Student(name, matric, year);

        John.registerSubject(subject);

        John.calculateFee();

    }

}

```

The following statements are wrong because the object is trying to call the member method before its creation.

Program 1.10 (Contains Error):

```

class obj {

    public static void main (String[] args ){

        John.displayAge();

```

```
    Student John = new Student("John Smith", 2345, 1969);  
  }  
}
```

Now consider the following program (Program 1.11). Do you think the program is correct?

Program 1.11:

```
class obj {  
    public static void main (String[ ] args ){  
        Student John = new Student("John Smith", 2345, 1969);  
        John.calculateFee( );  
        John.registerSubject("Java Programming");  
    }  
}
```

The above program is not so accurate because the method `calculateFee()` is called before the method `registerSubject()`. This does not make sense because student need to register the subject through the method `registerSubject()` before they can call the method `calculateFee()` because fee is calculated based on the number of subjects registered.

REMEMBER:

The sequence of methods calling by an object are very important. Objects need to call the methods according to the proper order.

For example, we cannot call method `registerSubject()` if an object `Student` has not done the `registerSemester`. To make a program more robust and tolerance for errors, it will be good idea to include codes in the member methods just to detect whether the method has been called prematurely. The inclusion of the statement `"if (registered==true)"` in the method `calculateFee()` in Program 5.8 is an example of this.

An object cannot call method that is not available in its class. The following statements in the Program 1.12 are wrong because the object John (which is an object of the class Student) is trying to call the method registerSemester() which is not exist in the class Student.

Program 1.12 (Contains Error):

```
class obj {  
  
    public static void main (String[] args ){  
  
        Student John = new Student("John Smith", 2345, 1969);  
  
        John.registerSemester(); /* Error: method registerSemester() does not exist  
                                   in the class Student */  
  
    }  
  
}
```

Like member method, we cannot call a constructor that does not exist in class. For example, Line 3 in the following program is an error because it is calling constructor method with one argument. You may recall that there is no constructor with one parameter in the class Student.

Program 5.15 (Contains Error):

```
Line 1      class obj1 {  
Line 2      public static void main (String[ ] args){  
Line 3          Student John = new Student("John Smith"); //ERROR  
Line 4      }  
Line 5      }
```

SUMMARY



- A **class** is a blueprint that defines the variables (or attributes) and the methods common to all objects of a certain kind.
- Components that make up a class are attributes, constructors and member methods
- Constructors used to initialize the attributes while member methods are used to perform operations on the attributes while attributes represent the state of the object
- There are some differences between constructor and member method in the aspects of the syntax and the nature of their tasks
- Object is an instance of class.
- Object is the most important thing in the object oriented paradigm.
- The keyword `new` is used to create an object from a class