Reading one – Summary notes

Benni E. [13-01-2021] SYSC2004

1. **Object wrappers:-** Object wrappers are OO code that has additional code inside of it. For example, consider loops being nested within an object. The object containing the loop is considered the “object wrapper”.
   - **1.1.** **Applications:-** Security, non-portable hardware features, etc.
2. **Object:-** An object has two components; attributes, and behaviours.
   - 2.1. **Attributes:-** Eye colour, height, age, etc.
   - 2.2. **Behaviours:-** Walking, talking, etc.
3. **Difference between OO and Procedural programming:-** In OOP the attributes and behaviours are within a single object. In procedural programming (or “structured”) the attributes and behaviours are separate.
4. **Data hiding:-** Data hiding is restricting access to certain attributes, or methods.
5. **Encapsulation:-** Encapsulation is the combining of attributes and methods, in the same entity. Encapsulation allows us to control access to the data in an object.
   - 5.1. **Example:-** When using a calculator, the sum method is accessed when you press the ‘+’ key. You then also pass your input to the calculator, whereby it sums them, without showing you how.
6. **Procedural programming:-** PP usually separates the data in a system from the operations which can manipulate the data.
   - **6.1.** **Example:-** If you want to send information across a network, only relevant data will be sent. We also expect the receiving-end to properly handle the data. Data in this context is often referred to as “packets”.
7. **OOP:-** In OOP, the data and the manipulative operations (the code) are both included, or encapsulated, in the object.
   - 7.1. **Example:-** When an object is transported across a network, the entirety of the object, data and all, is sent.
8. **Object data:-** The data stored within an objects is indicative of the state of the object. This data, inside the object is called attributes (i.e., age, height, gender, date-of-birth).
9. **Object behaviours:-** The behaviour of an object is indicative of what the object can do.
   - **9.1.** In PP the behaviour is defined by the procedures, functions, and subroutines of the programme.
   - **9.2.** In OOP the behaviours are contained within methods. You call a method by sending a message to it.
10. **Getters & Setters:-** A setter is a method which sets the attribute of an object to some value. A getter is a method which requests the attribute that was set by the setter. Getters can be

Reading one – Summary notes

Benni E. [13-01-2021] SYSC2004

remembered by their *return ...;* format. Getters and Setters are sometimes referred to as "accessor methods" and "mutator methods", respectively.

10.1. Getters and setters can also be properties of attributes.

a) Kindly refer to this example of getters and setters as properties of attributes:-

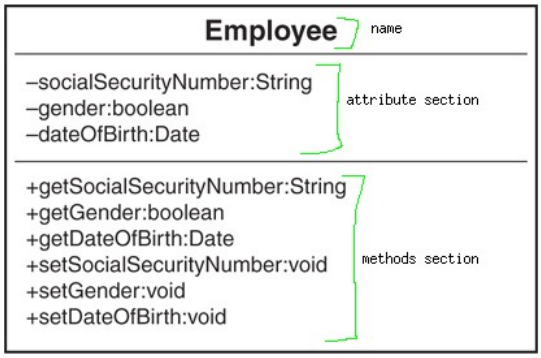
```
private string strName;

public String Name
{
    get { return this.strName; }
    set {
        if (value == null) return;
        this.strName = value;
    }
}
```

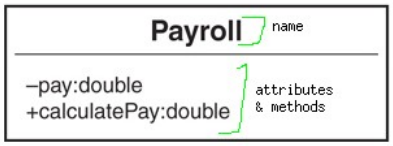
11. Interface of methods:- When focusing only on the interface of methods (not on the actual implementation) there are three things which we needs to know: the name of the method, the parameters passed to the method, the return type of the method.

12. **Class diagrams:-** A class diagram defines three sections of an object: the name, the attributes (data), the behaviours (methods). Kindly refer to the image below for a rough example (the + sign designates public, and the – designates private):-

Reading one – Summary notes

Benni E. [13-01-2021] SYSC2004





13. **Class:-** A class can be thought of as a blueprint for an object. When you initialize an object you use a class a prefix that defines how the object is built.

14. **Creating objects:-** Class are basically templates for objects. In other words, a class is used to create an object.

14.1. **Example:-**

```
public class Person{

    //Attributes
    private String name;
    private String address;

    //Methods
    public String getName(){
        return name;
    }
    public void set        Name(String n){
        name = n;
    }

    public String getAddress(){
        return address;
    }
    public void setAddress(String adr){
        address = adr;
    }

}
```

15. **Public:-** When a data type or method is defined as public, other objects can directly access, and modify it.

16. **Private:-** When a data type or method is defined as private, only that specific object can access, and modify it.

Reading one – Summary notes

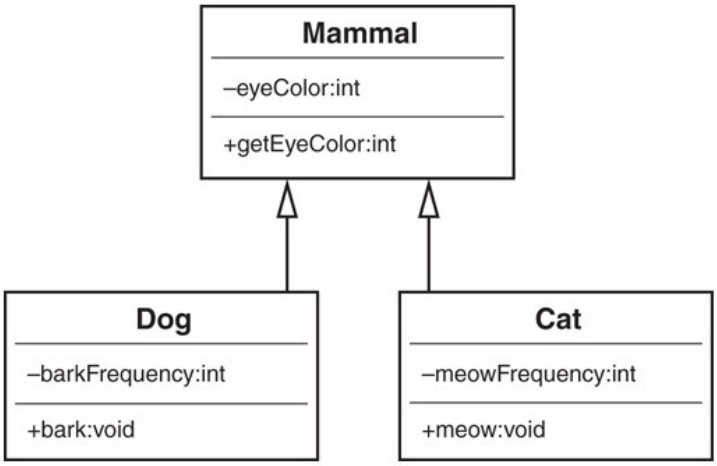
Benni E. [13-01-2021] SYSC2004

16.1. For data hiding to work properly, all attributes should be declared as private.

17. **Protected:-** When a data class is protected, only objects which are related to it can access, and modify it.

18. **Inheritance:-** Inheritance lets a class receive the attributes and methods of another class. This allows us to create new classes by abstracting out common attributes and behaviours from another class.

18.1. **Example:-** Kindly refer to the diagram below on inheritance. Note:- The Dog and Cat classes also have the attributes and methods from Mammal.



19. **Superclass:-** The superclass, parent class, or base class contains all the attributes that are common to the classes that inherit from it (i.e., the children classes). Recognise the Mammal class in the above example. Note:- The superclass does not necessarily have all of the attributes/ behaviours of its children/subclasses.

20. **Subclass:-** The subclass, child class, or derived class is an extension (or child) of the superclass. Recognise the Dog and Cat classes in the above example.

21. **Abstraction:-** A class can only have one parent, yet it can have many child classes. There are only two ways to build classes: inheritance, and composition. Inheritance allows one class to inherit from another class. Now recognise that we can abstract (pull) out attributes and behavious for common classes, from superclasses.

21.1. **Example:-** The relationship between a car and an engine can be used to illustrate abstraction. By separating the engine from the car we have obvious advantages. By building the engine separately, we can use the engine in more than just one car (along with other advantages). Although, we cannot say an engine is-a (22) car. Rather we would say that a car has a(n) (28) engine.

21.2. **Multiple inheritance:-** Classes in some languages such as C++ (not in Java) can have multiple parents, this is called single inheritance.

Reading one – Summary notes

Benni E. [13-01-2021] SYSC2004

21.3. **Single inheritance:-** Classes in other languages such as Java have a more strict version of inheritance, where classes can only have one parent. This is called single inheritance.

22. **Is-a relationship:-** Any relationship where the subclasses have the same attributes and behaviours as the superclass.

23. **Polymorphism:-** A Greek word meaning “many shapes”. It is closely related to inheritance. It is also considered one of the most notable advantages to OOP. Polymorphism simply explains that each class is able to respond differently to the same superclass method.

24. **Abstract:-** When a method is defined as abstract, a subclass must provide the implementation for this method.

24.1. **Example:-** Kindly refer to this example. Notice the subclass has the abstract method defined, and thus Shape requires the subclass to provide a getArea().

```
public abstract class Shape{

    private double area;

    public abstract double getArea();

}
```

25. **Constructor:-** A constructor is the entry point for the class, where the object is built; the constructor is a good place to perform initiliazations and beginning tasks. A class can be recognised by a missing return type, along with a method of the same name as the superclass.

25.1. **Calling a constructor:-** When calling a constructer you will see a piece of code similar to: “Cabbie myCabbie = new Cabbie();”. The *new* keyword creates a new instance of the Cabbie class, and allocates the memory required. The constructer itself is then called, *Cabbie(...);* passing any arguments in the parameter list. The new cobbie object being created by the *Cabbie(…);* is the constructor.

25.2. **Missing a constructor:-** If a constructor is missing in the code, the compiler will automatically insert the constructer of the superclass. Generally you should always provide a constructor.

26. **Stack:-** A stack is LIFO (Last In, First Out). Popping from a LIFO means removing the most recently placed value (think of a plate dispenser in a restaurant).

27. **Composition:-** Composition is the idea that objects are often composed of other objects.

Reading one – Summary notes

Benni E. [13-01-2021] SYSC2004

28. **Has-a relationship:-** Composition relationships are considered has-a relationships, whereas inheritance have is-a relationships.

29. **Comment styles in Java:-** There are two types of comment styles in Java:-

29.1. **/* ...comment… */:-** This type of comment style can span more than one line.

29.2. **// ...comment...:-** This type of comment renders the whole line as a comment, thus it only spans one line.

30. **Attributes:-** Attributes store information about an object.

30.1. **Example:-** This attribute stores the name of a company:

```
private static String companyName = "Blue Cab Company";
```

- The keywords here are *private* and *static*. Private signifies that a method or variable can be accessed only within the declaring object. Static keyword signifies that there will only be one copy of this attribute for all the object instantiated by this class (basically this is a class attribute).

31. **References:-** When a reference to an object is created it does not meant that memory has been allocated for said object.

31.1. **Example:-** This attribute is a reference to another object. The class, *Cab*, holds information about a specific cab, such as its serial, maintenance records, etc.:

```
private Cab myCab;
```

- As mentioned above, this does not allocate memory for the *myCab* object. Although, it is likely that the Cab object was in fact created by another object. Thus, the object reference would be passed to the *Cabbie* object. We are not interested in the code surround *Cab*.

32. **Null:-** The value of *Null* represents a value of nothing, not zero, just nothing ;-(. This is often used to check whether or not an attribute has been properly set. In some languages this is not permitted with the *string* type (i.e., in .NET).