# Advanced Higher

# Object Oriented Programming

# Theory Questions & Answers

The underlying principles of Object Oriented Programming and may be summarised as follows:

**Objects**: represent the basic building block.

**Class**: the blueprint of an object.

**Sub-class**: a new class that has been derived from an existing class

**Super-class**: the existing class from which a class has been derived.

**Property**: the data that is held within the object.

**Method**: the functions and procedures to operate on the properties within the object – these may be classified as **Constructors** (including **Getters** and **Setters**). As the names indicate, a getter method retrieves an attribute, and a setter method changes it.

**Encapsulation**: the mechanism of preventing a programmer from getting direct access to the properties or to the code of the methods. To encapsulate something means to enclose it in some kind of container. In OOP, the term encapsulation means bundling data and the methods that work on that data within one component - i.e. an object.

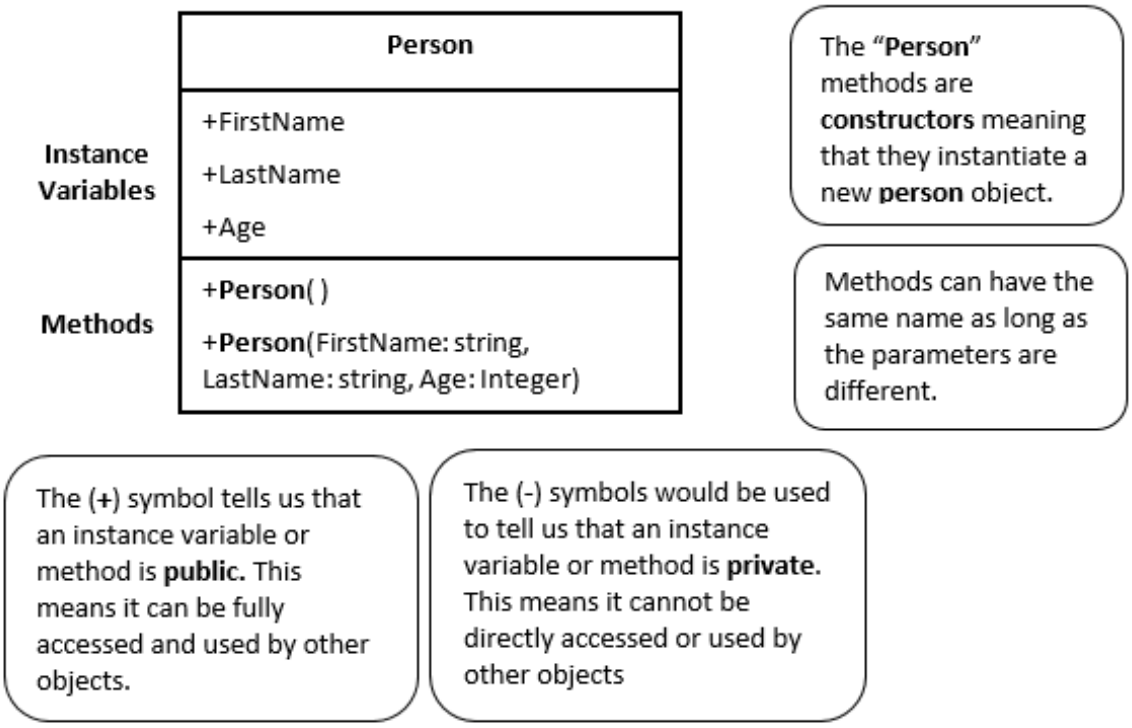
**Inheritance**: the mechanism of making a new sub-class from an existing one (the super-class), passing to the new sub-class the properties and methods from the existing super-class in the process.

**Instantiation**: creating an object from a class.

**Polymorphism**: being able to create instances from different classes that share the same super-class so on the surface may seem different, but will have some of their methods and properties in common. The word polymorphism means having 'many forms'. In programming, polymorphism is the ability to use the same interface (function) for different forms of data. In simple terms, this permits functions to be applied to objects of different types at different times.

## Classes

The **UML diagram** below shows a ***Person*** class which contains three instance variables and two constructor methods.




***SQA Reference Language: Defining a class***

**CLASS *person* IS {STRING FirstName, STRING LastName, INTEGER Age}**

**METHODS**

**CONSTRUCTOR (STRING first_name, STRING last_name, INTEGER age)**

**DECLARE THIS.*FirstName* INITIALLY *first_name***
**DECLARE THIS.*LastName* INITIALLY *last_name***
**DECLARE THIS.*Age* INITIALLY *age***

**END CONSTRUCTOR**

**END CLASS**

**Note:** "This" is used to access (refer to) the object which invoked the constructor method.

**Instances**

An object is an instance of a class. Creating a new object is called instantiating.

There are three steps to instantiation:

- **Declaration:** A variable declaration with a variable name with an object type.
- **Instantiation:** The 'new' key word is used to create the object.
- **Initialisation:** The 'new' keyword is followed by a call to a constructor. This call initialises the new object.

Instantiating a new ***Person*** object using this class with specific values is done using the code below:

```
public static void main(String[] args) {

            Person p1 = new Person("Adam", "Jones", 24);

}
```

Here, the variable p1 is being **declared**. p1 is **instantiated** as an Person object using the new keyword. Finally, the constructor method is used to **initialise** the object using three instance variables.

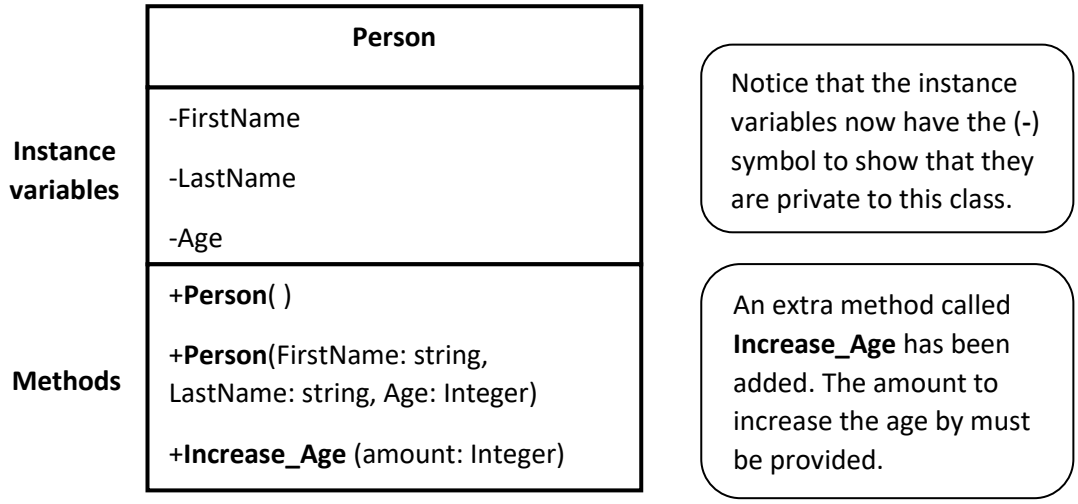
***SQA Reference Language: Instantiating***

DECLARE ***fred*** INITIALLY ***person*** **{“Fred”, “Black”, 32}**

**Encapsulation**

To prevent unregulated altering, it is better to make instance variables private to hide them from other objects, and provide additional methods for making changes to the data items.

Other objects can only make changes to private data items using the methods provided. This is the purpose of **encapsulation**.




```
public Class Person{

  private String firstname;
  private String lastname;
  private int age;


   public Person(){
      this.firstname = "";
      this.lastname = "";
      this.age = 0;
      }

   public Person(String fn, String ln, int a){
        this.firstname = fn;
        this.lastname = ln;
        this.age = a;
       }


   public void increaseAge(int amount){

        this.age = this.age + amount;

    }
`
```

The ***Increase_Age*** method means that the only way of updating the **age** data item is to increase it by a specified amount. At the moment there is **no method** provided for reducing the age so it cannot be done.

*Note: **void** means that the method has no return (output) value.*

To increase the age of a person, the following code would be used:

```
public static void main(String[] args) {

	Person p1 = new Person("Adam", "Jones", 24);
	p1.IncreaseAge(15);

}
```

More methods can now be added to the Person class to allow instance variable values to be changed or viewed.

```
public void set_firstname(String first_name){
	this.firstname = first_name;
}

public void set_lastname(String last_name){
	this.lastname = last_name;
}

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

**Setter** methods like those above, set the values of data items to a provided value.

```
public String get_firstname(){
	return firstname;
}


public String get_lastname(){
	return lastname;
}


public int get_age(){
	return age;
}
```

**Getter** methods like those above, return the value of a data item.

Using getters and setters would work like this:

```
p1.set_firstname("Bob");
p1.set_lastname("Black");
p1.set_age(17);


System.out.println(p1.get_firstname() + " " + p1.get_lastname());

System.out.println(p1.get_age());
```

***Fred.set_age(29)***

***Fred.get_age***

**Note:** the getter and setter methods above are invoked by the object, “Fred”.

**SET *Fred.Age* TO *29***

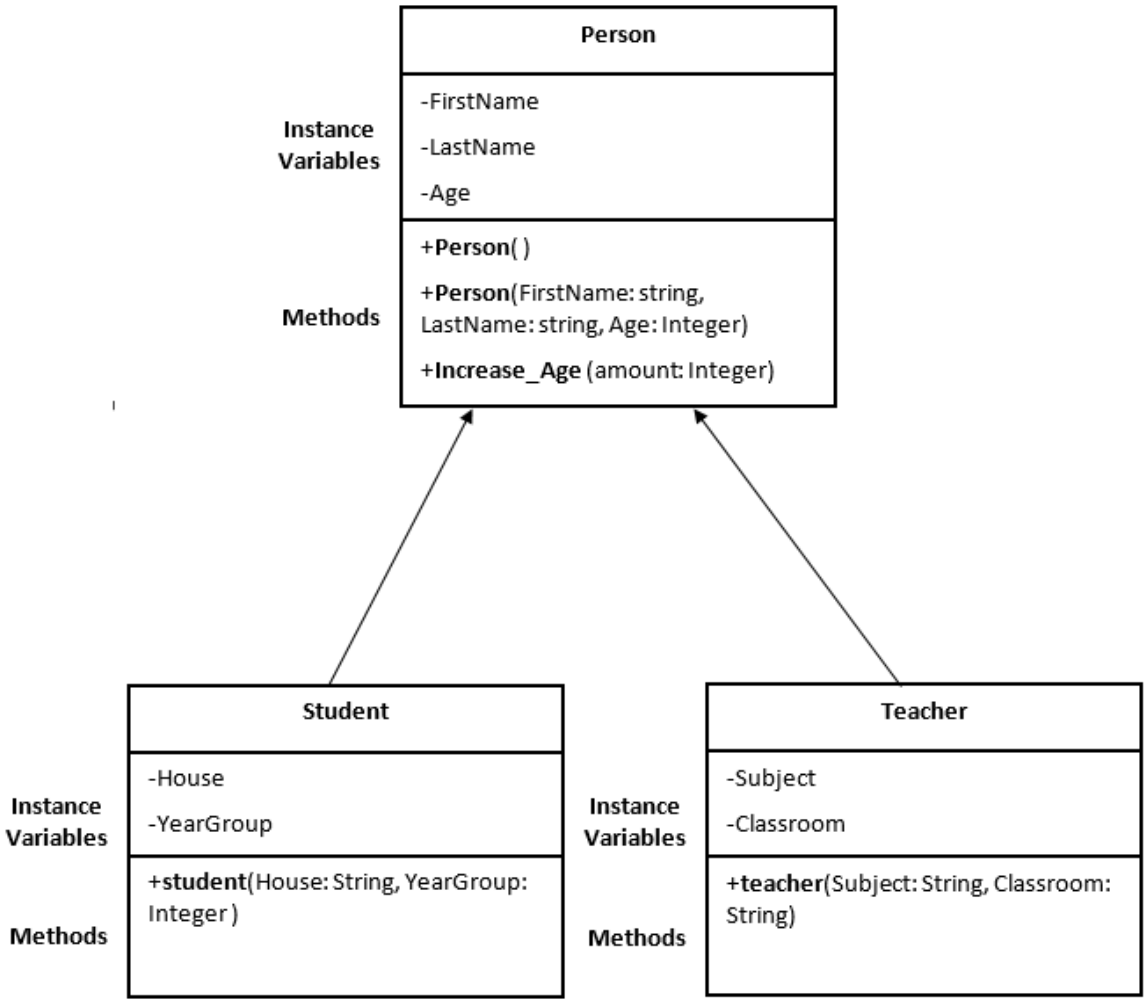
**SET *Fred.FirstName* TO *“John”***

**Note:** the above code is **not allowed** because it tries to access Age and FirstName directly which are both private. Encapsulation requires that a method is used to alter data items.

**Inheritance**

Inheritance means that when a **subclass** is created, it copies all of the data items and methods from the **superclass**.

Students and teachers are both subclasses of the person class. Students and teachers both have a first name, last name an age, but they also have unique properties of their own.




The arrows show that Student and Teacher inherit all of the data items and methods from the person class.

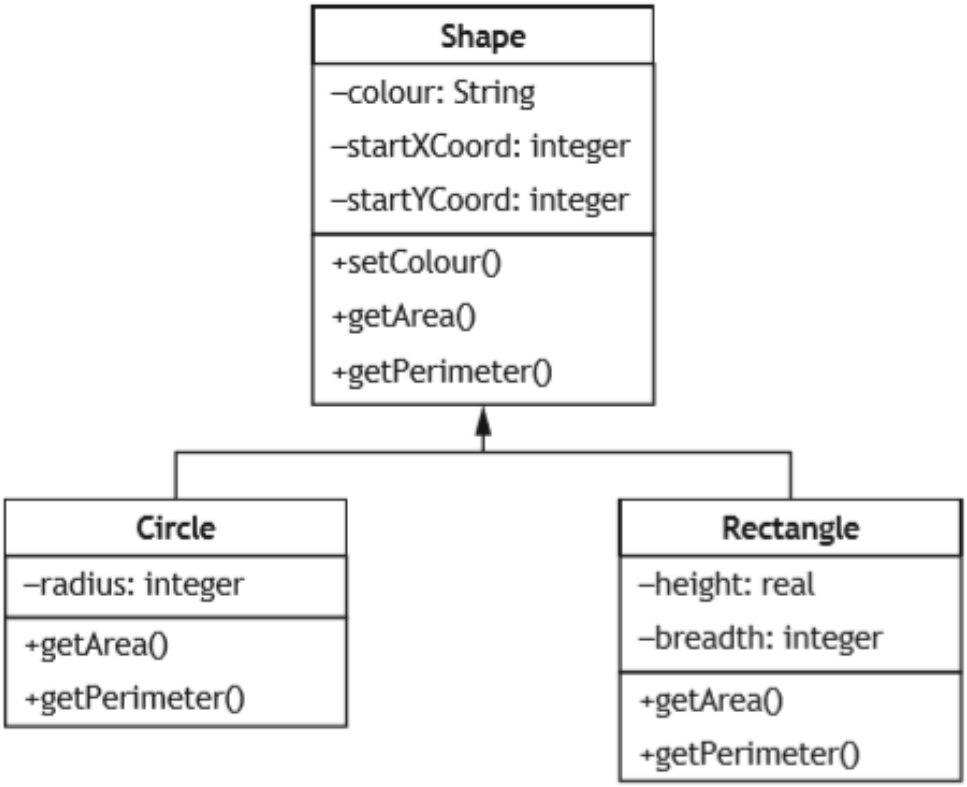
***SQA Reference Language: inheritance***

```
CLASS student INHERITS person WITH {STRING House, INTEGER yearGroup}

METHODS

    CONSTRUCTOR (STRING first_name, STRING last_name, INTEGER age,
        STRING house, INTEGER year_Group)

        DECLARE  THIS.FirstName INITIALLY first_name
        DECLARE  THIS.LastName INITIALLY last_name
        DECLARE  THIS.Age INITIALLY age
        DECLARE  THIS.House INITIALLY house
        DECLARE  THIS.yearGroup INITIALLY year_Group

    END CONSTRUCTOR

END CLASS
```

**Note:** the above class inherits person and adds house and yearGroup. All five data items can then be initialised in the constructor method.

Exam style – Example Questions

# Example 1




Using appropriate object-oriented terminology, explain why the following statement would be invalid.

```
SET shape1.colour TO "blue"
```

2

The `getArea()` methods of the `Shape` and `Rectangle` classes are shown below.

Shape class

```
FUNCTION getArea() RETURNS REAL
    SET area TO 0.0
    RETURN area
END FUNCTION
```

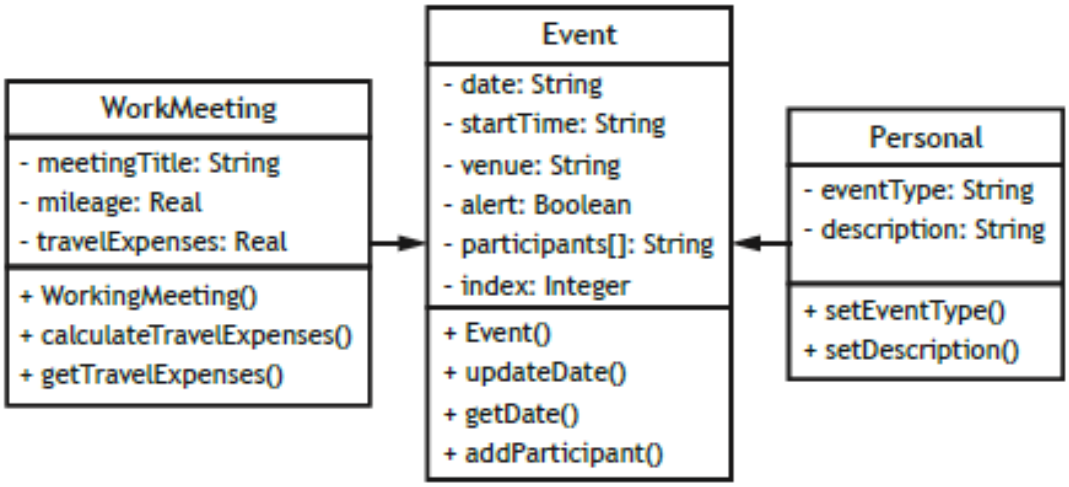
Rectangle class

```
OVERRIDE FUNCTION getArea() RETURNS REAL
    SET area TO THIS.length * THIS.breadth
    RETURN area
END FUNCTION
```

Using appropriate object-oriented terminology, explain the use made of the `OVERRIDE` statement in the `getArea()` method of the `Rectangle` class. 2

| Expected response | Max mark | Additional guidance |
|---|---|---|
| Encapsulation has been used to restrict access to the `colour` variable.<br><br>This variable can only be accessed using the `setColour()` method of the `Shape` class. | 2 | 1 mark for encapsulation or use of private property.<br><br>1 mark for method required to edit the contents of the variable. |
| The `OVERRRIDE` statement is used to redefine the `getArea()` method that was inherited from the `Shape` class. | 2 | 1 mark for use of `OVERRIDE`.<br><br>1 mark for inherited method. |

## Example 2




```
Line 1   CLASS Event IS { STRING date, STRING startTime, STRING venue,
         BOOLEAN alert, ARRAY OF STRING participants, INTEGER index }

Line 2     METHODS
Line 3       CONSTRUCTOR ( STRING date, STRING startTime, STRING venue,
             BOOLEAN alert )
Line 4         DECLARE THIS.date INITIALLY date
Line 5         DECLARE THIS.startTime INITIALLY startTime
Line 6         DECLARE THIS.venue INITIALLY venue
Line 7         DECLARE THIS.alert INITIALLY alert
Line 8         DECLARE THIS.participants INITIALLY [NULL] * 20
Line 9         DECLARE THIS.index INITIALLY 0
Line 10      END CONSTRUCTOR

Line 11      PROCEDURE updateDate(STRING eventDate)
Line 12        SET THIS.date TO eventDate
Line 13      END PROCEDURE

Line 14      FUNCTION getDate() RETURNS STRING
Line 15        RETURN THIS.date
Line 16      END FUNCTION

Line 17      PROCEDURE addParticipant(STRING name)
Line 18        SET THIS.participants[index] TO name
Line 19        SET index TO index + 1
Line 20      END PROCEDURE

Line 21  END CLASS
```

## Sample Questions

Using a programming language of your choice, write the code equivalent to Line 1 of the `Event` class, for the declaration for the `WorkMeeting` class. 2

Describe the use of the `Constructor` method in the `Event` class. 1

(i) Using appropriate object-oriented terminology, explain the operation and effect of executing the following code. 2

```
DECLARE sales INITIALLY WorkMeeting ( "22/6/19",
"14:00", "Head Office", false, "June sales meeting",
12.0, 0.0 )
```

(ii) Explain the effect of executing the following code. 2

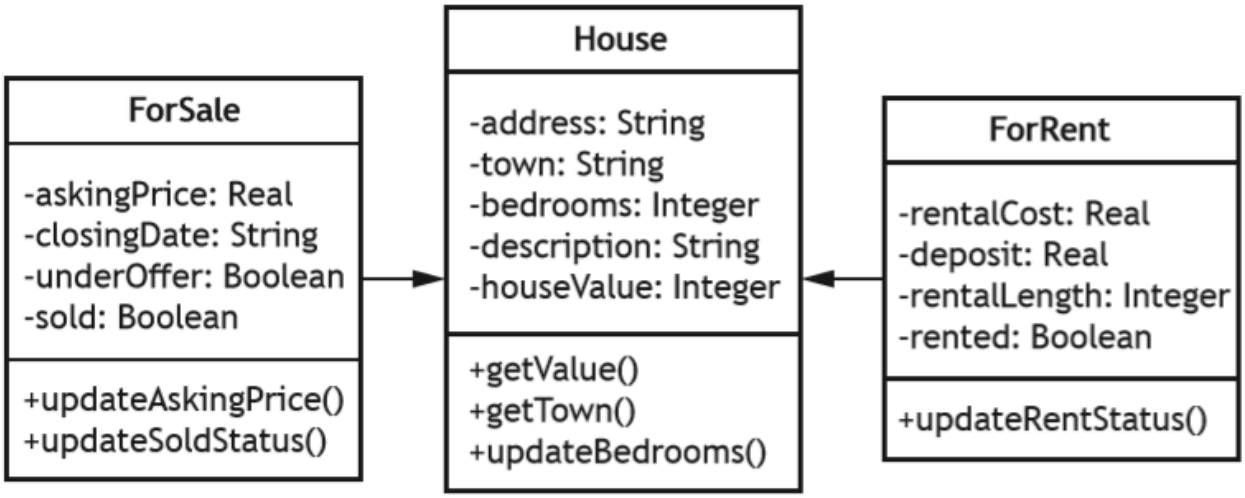
```
sales.addParticipant("Chao Li")
```

(iii) The venue for the event called `sales` has been changed.

By referring to the UML class diagram and using appropriate object-oriented terminology, explain why it is not possible to edit the value already assigned to the `venue` property. 2

| on | Expected response | Max mark | Additional guidance |
|---|---|---|---|
| | `CLASS WorkMeeting INHERITS Event WITH { STRING meetingTitle, REAL mileage, REAL travelExpenses }` | 2 | 1 mark for class declaration indicating use of inheritance<br><br>1 mark for additional attributes; only award mark for nine attributes if no inheritance indicated |
| | The constructor assigns initial values upon instantiation of an object. | 1 | Description must mention creation/instantiation of an object. |
| (i) | A new object of the `WorkMeeting` subclass has been instantiated (1 mark)<br>The values are assigned to the relevant instance variables with the constructor initialising an empty array to store the list of participants and initialising the array index as zero (1 mark) | 2 | Award 1 mark each for 2 of the following:<br>• instantiation/creation of object or invoke constructor method for WorkMeeting class<br>• assignment using the 7 values provided and additional two attributes of Event class |
| (ii) | Award 1 mark each for any two of the following:<br>• A new participant called Chao Li has been added to the array of meeting participants<br>• and the array index is incremented<br>• Invoke addParticipants() method | 2 | |
| (iii) | Encapsulation means that a property cannot be edited directly. (1 mark)<br>A method needs to exist in order to update the venue property. (1 mark) | 2 | 1 mark for encapsulation/private variable<br><br>1 mark for need for method |

**Example 3**




```
Line 1   CLASS House IS {STRING address, STRING town, INTEGER
         bedrooms, STRING description, INTEGER houseValue}

Line 2   METHODS

Line 3       CONSTRUCTOR (STRING address, STRING town, INTEGER
             bedrooms, STRING description, INTEGER houseValue)
Line 4           DECLARE THIS.address INITIALLY address
Line 5           DECLARE THIS.town INITIALLY town
Line 6           DECLARE THIS.bedrooms INITIALLY bedrooms
Line 7           DECLARE THIS.description INITIALLY description
Line 8           DECLARE THIS.houseValue INITIALLY houseValue
Line 9       END CONSTRUCTOR

Line 10      PROCEDURE updateBedrooms (INTEGER noOfBedrooms)
Line 11          SET THIS.bedrooms TO noOfBedrooms
Line 12      END PROCEDURE

Line 13      FUNCTION getTown() RETURNS STRING
Line 14          RETURN THIS.town
Line 15      END FUNCTION

Line 16      FUNCTION getValue() RETURNS INTEGER
Line 17          RETURN THIS.houseValue
Line 18      END FUNCTION

Line 19  END CLASS
```

## Sample Questions

Using appropriate object-oriented terminology, explain the operation and effect of executing the following code.

(i) `DECLARE saleHouse1 INITIALLY ForSale ("12 Albert Road", "Dundee", 2, "Well presented bungalow", 140000, 150000, "20/6/18", FALSE, FALSE)` 2

(ii) `saleHouse1.updateBedrooms(3)` 2

Using appropriate object-oriented terminology, explain why the following statements would be invalid.

(i) `SET saleHouse1.description TO "Well presented house with garden."` 2

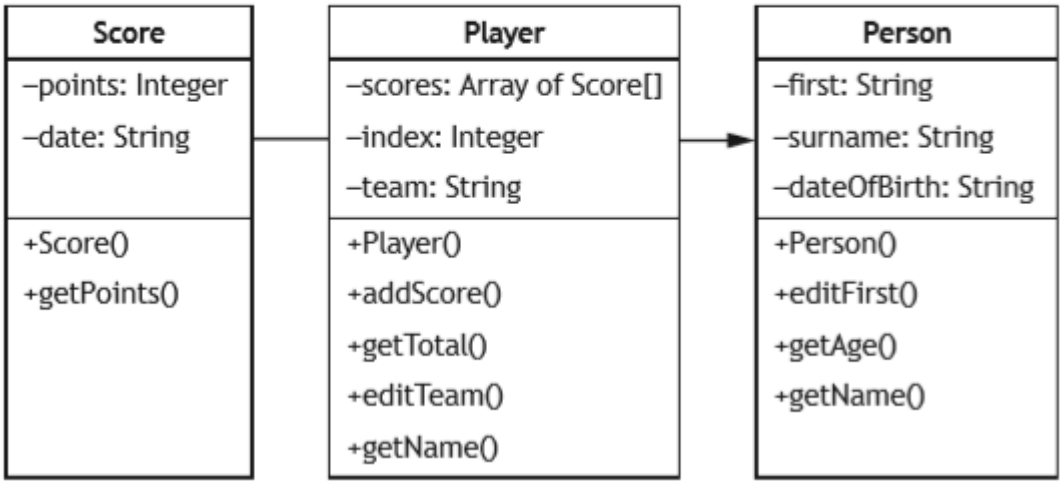
(ii) `saleHouse1.updateRentStatus(TRUE)` 1

| | | | |
|---|---|---|---|
| (i) | Instantiates an object (called saleHouse1) that belongs to the `ForSale` class (with given values/actual parameters). | 2 | 1 mark each for any 2 of the following:<br>• instantiates/create object<br>• class or using values/parameters<br>• constructor method invoked. |
| (ii) | The procedure call invokes the `updateBedrooms` method of the `saleHouse1` object (with the value/actual parameter 3).<br><br>The method `updateBedrooms` is inherited from `House` superclass.<br><br>A value of 3 is assigned to the property `bedrooms` (in the `saleHouse1` object). | 2 | 1 mark each for any 2 of the following:<br>• invoking method (or procedure)<br>• use made of inheritance<br>• updating property. |
| (i) | Encapsulation has been used to restrict access to the description variable/property.<br><br>This property can only be accessed through methods in `ForSale` class (or inherited methods from `House` class). | 2 | 1 mark for encapsulation or use of private property.<br><br>1 mark for method required to use/change property. |
| (ii) | Since the `updateRentStatus` method is not part of `saleHouse1` object.<br><br>OR<br><br>The `updateRentStatus` method is only available to objects that belong to the `ForRent` class. | 1 | |

## Example 4

An object-oriented program is used to store and return statistics about basketball teams and their players.

A simplified version of the UML class diagram for the program is shown below.




Some of the program code is shown below.

```
...
Line 20    CLASS Person IS { STRING first, STRING surname,
           STRING dateOfBirth }

Line 21    METHODS

Line 22       CONSTRUCTOR ( STRING first, STRING surname,
                            STRING dateOfBirth )
Line 23          DECLARE THIS.first INITIALLY first
Line 24          DECLARE THIS.surname INITIALLY surname
Line 25          DECLARE THIS.dateOfBirth INITIALLY dateOfBirth
Line 26       END CONSTRUCTOR

Line 27       PROCEDURE editFirst( STRING first )
Line 28          SET THIS.first TO first
Line 29       END PROCEDURE
...
...

Line 35       FUNCTION getName() RETURNS STRING
Line 36          DECLARE outputPhrase AS STRING
Line 37          SET outputPhrase TO "My name is " & THIS.first
                 & " " & THIS.surname & " and I like basketball"
Line 38          RETURN outputPhrase
Line 39       END FUNCTION

Line 40    END CLASS
```

```
...
Line 60    CLASS Player INHERITS Person WITH { ARRAY OF Score
           scores, INTEGER index, STRING team }

Line 61    METHODS

Line 62       CONSTRUCTOR ( STRING first, STRING surname,
              STRING dateOfBirth, STRING team )
Line 63          DECLARE THIS.first INITIALLY first
Line 64          DECLARE THIS.surname INITIALLY surname
Line 65          DECLARE THIS.dateOfBirth INITIALLY dateOfBirth
Line 66          DECLARE THIS.team INITIALLY team
Line 67          DECLARE THIS.scores AS ARRAY OF Score INITIALLY
                 []
Line 68          DECLARE THIS.index INITIALLY 0
Line 69       END CONSTRUCTOR
...
...
Line 101   DECLARE person1 AS Person INITIALLY ( "Wayne",
           "Nowitzki", "19/06/1990" )
Line 102   person1.editFirst( "Dwayne" )
Line 103   SEND person1.getName() TO DISPLAY
...
```

(a) Using appropriate object-oriented terminology, explain the operation and effect of

(i) Line 101 of the program **2**

(ii) Line 102 of the program. **2**

(b) State the output generated by Line 103 of the program. **1**

(c) (i) Amanda Greene was born on 26th April 1987 and she has signed up to play basketball for the Burnside Braves.

Using a programming language of your choice, write the line of code needed to instantiate a `Player` object called `player2` to store Amanda's details. **1**

(c) (ii) The following line of code is added to the program.

```
player2.getName()
```

The output produced by the program is shown below.

```
My name is Amanda Greene and I play basketball for the Burnside Braves.
```

Explain the meaning of the term polymorphism by making reference to the `getName()` methods in this program. **2**

(d) The `addScore()` method of the `Player` class is shown below.

```
Line 70   PROCEDURE addScore(newPoints, when)
Line 71     SET result TO Score INITIALLY (newPoints, when)
Line 72     SET THIS.scores[THIS.index] TO result
Line 73     SET THIS.index TO THIS.index + 1
Line 74   END PROCEDURE
```

Using appropriate object-oriented terminology, explain the operation and effect of the following line of code. **3**

```
player2.addScore(23, "27/01/2019")
```

(e) The `getTotal()` method of the `Player` class is used to calculate and return the total score for an individual player.

Using a programming language of your choice, write the code needed to implement the `getTotal()` method. **3**

| uestion | | Expected response | Max mark | Additional guidance |
|---|---|---|---|---|
| (a) | (i) | A new object of the `Person` class has been instantiated<br><br>**AND**<br><br>populated with a value for each instance variable in the class. | **2** | 1 mark for instantiation of an object of `Person` class.<br><br>1 mark for assignment of values to instance variables of the `Person` class. |
| | (ii) | This code is used to invoke the `editFirst()` method of the `person1` object (with the value/actual parameter 'Dwayne').<br><br>Changes the value of the instance variable `first` (in the `person1` object) to Dwayne. | **2** | 1 mark for invoking `editFirst()` method.<br><br>1 mark for updating the instance variable `first` of the `person1` object. |
| (b) | | My name is Dwayne Nowitzki and I like basketball. | **1** | |
| (c) | (i) | `SET player2 AS Player INITIALLY ("Amanda", "Greene", "26/04/1987", "Burnside Braves")` | **1** | All four values must be used correctly. |
| | (ii) | Polymorphism refers to the ability to redefine the `getName()` method of the `Player` subclass that has been inherited from the `Person` super class. The inherited code is overwritten meaning that the subclass and class respond differently to any message received. | **2** | 1 mark for ability to redefine the inherited `getName()` method.<br><br>1 mark for ability to respond differently to the same message. |
| (d) | | A new element (consisting of a score and date) is added to the array of `Score` objects that belongs to the `Player` object called `player2`.<br><br>The index of the `scores` array (the array of `Score` objects) is incremented by adding one to the class variable `numberGames` that belongs to the `Player` class. | **3** | 1 mark for correct use of the array of `Score` objects in explanation.<br><br>1 mark for correct relationship between `Score` and `Player` objects used in explanation.<br><br>1 mark for appropriate reference to indexing of the `scores` array by incrementing the class variable. |

| Question | | | Expected response | Max mark | Additional guidance |
|---|---|---|---|---|---|
| 8. | (e) | | `FUNCTION getTotal() RETURNS INTEGER`<br><br>`  DECLARE totalScore`<br>`  INITIALLY 0`<br><br>`  REPEAT FROM 0 TO`<br>`  THIS.index - 1 DO`<br>`  SET totalScore TO`<br>`  totalScore +`<br>`  scores.getPoints[THIS`<br>`  .index]`<br>`  END REPEAT`<br><br>`  RETURN totalScore`<br><br>`END FUNCTION` | 3 | 1 mark for initialising and correctly updating the local variable `totalScore`.<br><br>1 mark for loop making correct use of the instance variable `index` and returning the calculated `totalScore`.<br><br>1 mark for correct use of `getTotal()` method of the `scores` array. |

# SQA Provided Notes

To model a system, it is important to capture the static behaviour of the system.

A class diagram is used for a quick overview of the system. It describes the structure of a system by showing its:

classes
variables, structures and types
methods of the class
relationships between the classes

The purpose of a class diagram is to model the static aspect of the system.

## Drawing a class diagram

A class is a blueprint for an object. A class diagram describes each class and the relationships between the classes.

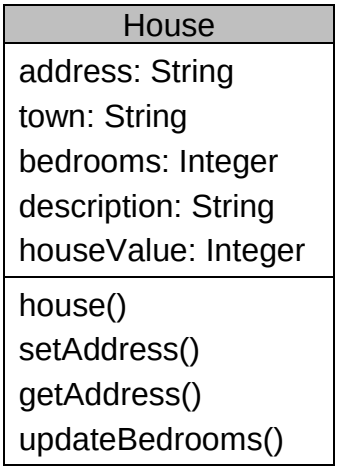
### UML class notation

A class diagram consists of:

a class name
instance variables and data types:
    public
    private
methods:
    public
    private
    constructor
inheritance between classes

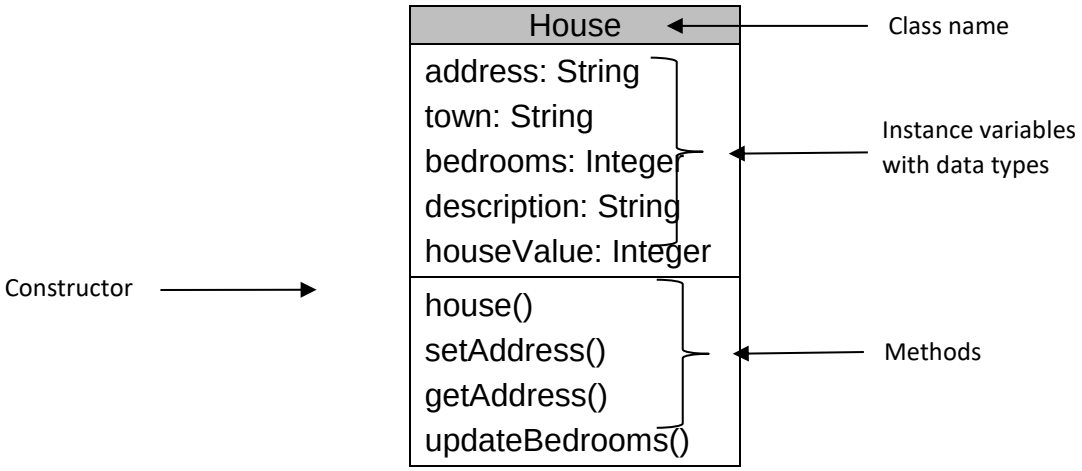
**Example**
A program is being written for an estate agency to store the details of houses for sale or available to rent.

**Class diagram for House**
Part of the class diagram for the House class is shown below.




**Explanation**
The class diagram indicates the:

class name
instance variables with data types in the class (instantiation variables)
methods associated with the class (including the constructor method)




**Constructor**
A constructor is shown on a UML class diagram in the methods section. The constructor will have the same name as the class name. The constructor method is used to create an individual object that belongs to the class.

**Public and private**
The instance variables and methods within a class can be public or private elements.

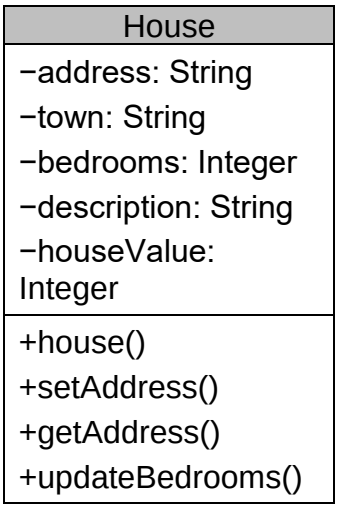
Public elements can be used by any class; however, private elements can only be used by the owning class.

UML allows any variable or method to be shown as public or private.

In a class diagram:

public elements are preceded with a + sign
private elements are preceded with a − sign

The House class, with public and private elements, will look as follows.




The set and get methods (sometimes called mutators and accessors) are needed to retrieve (get) or edit (set) the values held in private variables.

**Example code: `setAddress()`**

Used to edit the value stored in the private instance variable `address`.

```
PROCEDURE setAddress(STRING newAddress)
      SET THIS.address TO newAddress
END FUNCTION
```

**Example code: `getAddress()`**

Used to retrieve the value stored in the private instance variable `address`.

```
FUNCTION getAddress() RETURNS STRING
      RETURN THIS.address
END FUNCTION
```

**Inheritance**

UML allows the object-oriented construct of inheritance to be exemplified.

A subclass can inherit all of the properties and methods of a superclass.

On a UML class diagram, this type of inheritance is indicated by an arrow from the subclass to the superclass.

### Array of objects

The instance variables of a class or subclass can include an array data structure. This can be used to store instances of another class.

An array of objects is written as:

```
scores: Array of Score[ ]
```

where Score is another class. On a UML class diagram, the connection between the array of objects and the object (class) is also indicated by an arrow.

#### Example

The program below is for an estate agency to store the details of houses available for sale or to rent.