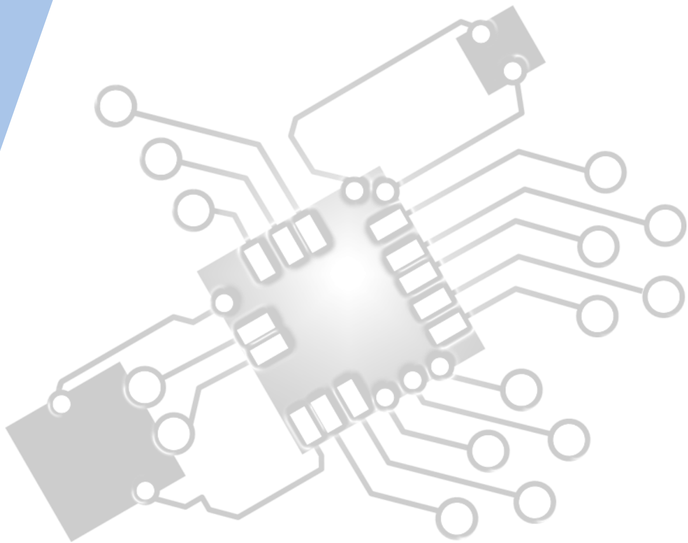
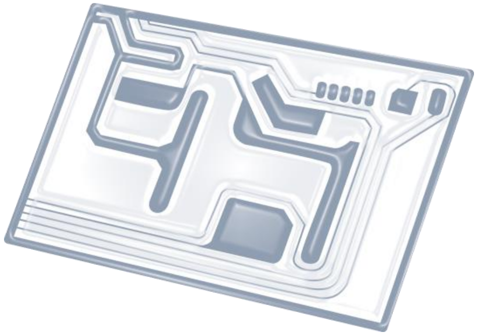
Objects as a programming concept

IB Computer Science



Content developed by
Dartford Grammar School
Computer Science Department

HL Topics 1-7, D1-4



1: System design

2: Computer Organisation

3: Networks

4: Computational thinking

5: Abstract data structures

6: Resource management

7: Control

D: OOP

Content developed by Dartford Grammar School Computer Science Department

HL & SL D.1 Overview

D.1 Objects as a programming concept

D.1.1 Outline the general nature of an object

D.1.2 Distinguish between an object (definition, template or class) and instantiation

D.1.3 Construct unified modelling language (UML) diagrams to represent object designs

D.1.4 Interpret UML diagrams

D.1.5 Describe the process of decomposition into several related objects

D.1.6 Describe the relationships between objects for a given problem

D.1.7 Outline the need to reduce dependencies between objects in a given problem

D.1.8 Construct related objects for a given problem

D.1.9 Explain the need for different data types to represent data items

D.1.10 Describe how data items can be passed to and from actions as parameters

1: System design

2: Computer Organisation

3: Networks

4: Computational thinking

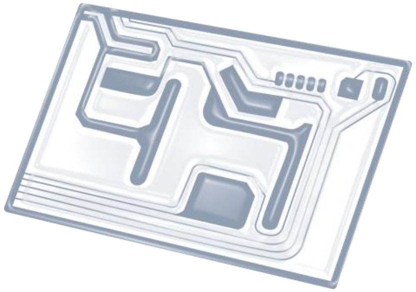
5: Abstract data structures

6: Resource management

7: Control

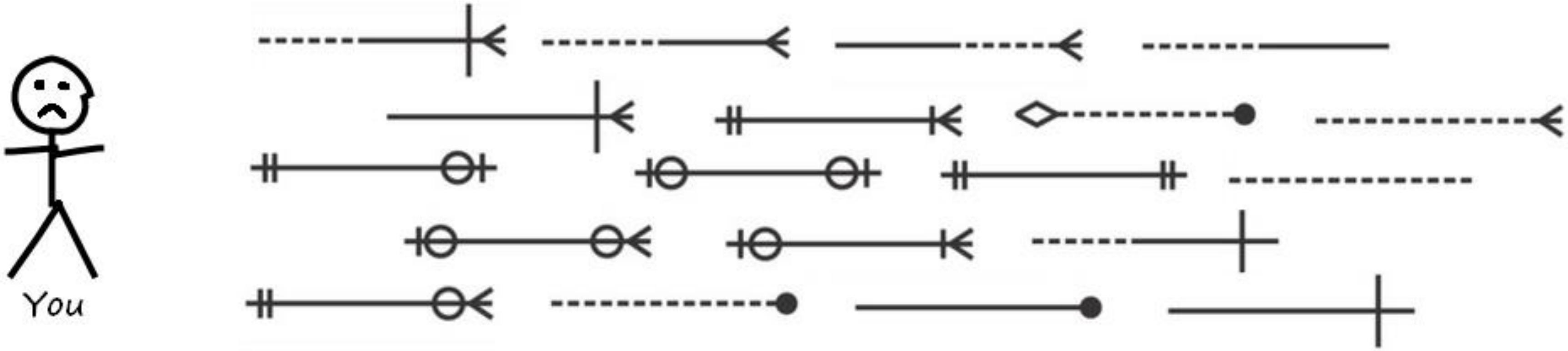
D: OOP

Content developed by Dartford Grammar School Computer Science Department



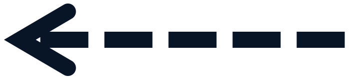
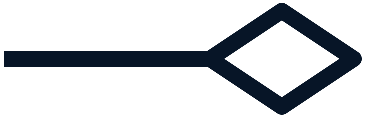
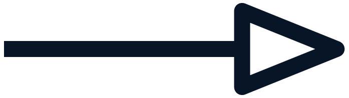
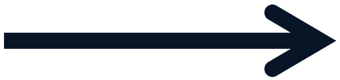
Topic D.1.6

Describe the **relationship between objects** for a given problem



Four types of relationships

There are four main types of relationships between objects:

- **Dependency** – *“uses”* 
- **Aggregation** – *“has a”* 
- **Inheritance** – *“is a”* 
- **Association** – “uses” 

Comparison

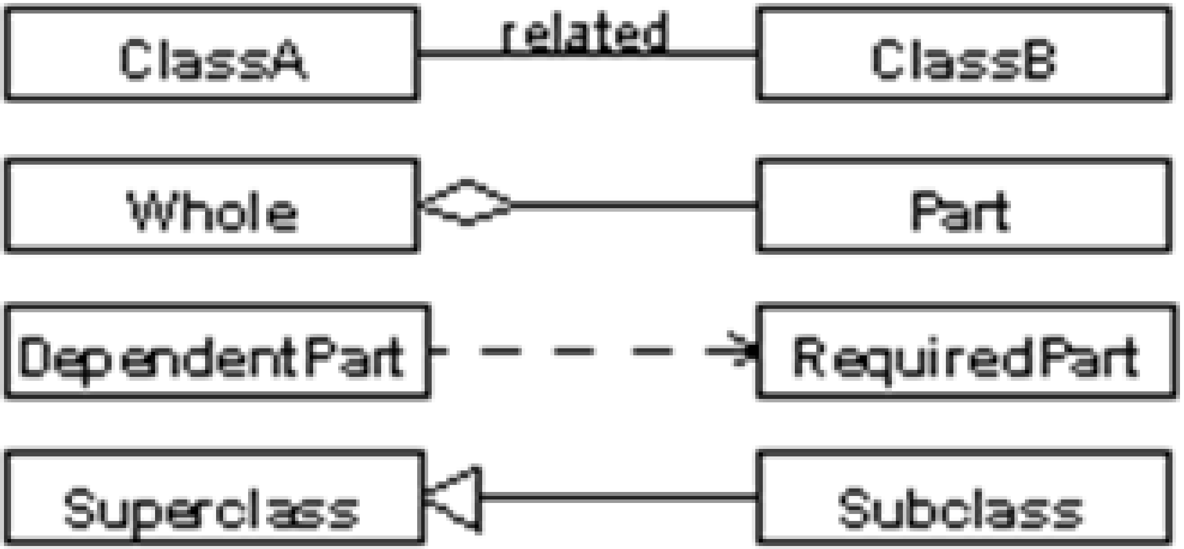
Generally speaking, **Association** is the most generic relationship. The other three are more specific and are used in particular situations.

Association

Aggregation

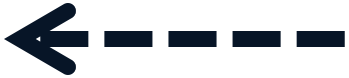
Dependency

Inheritance



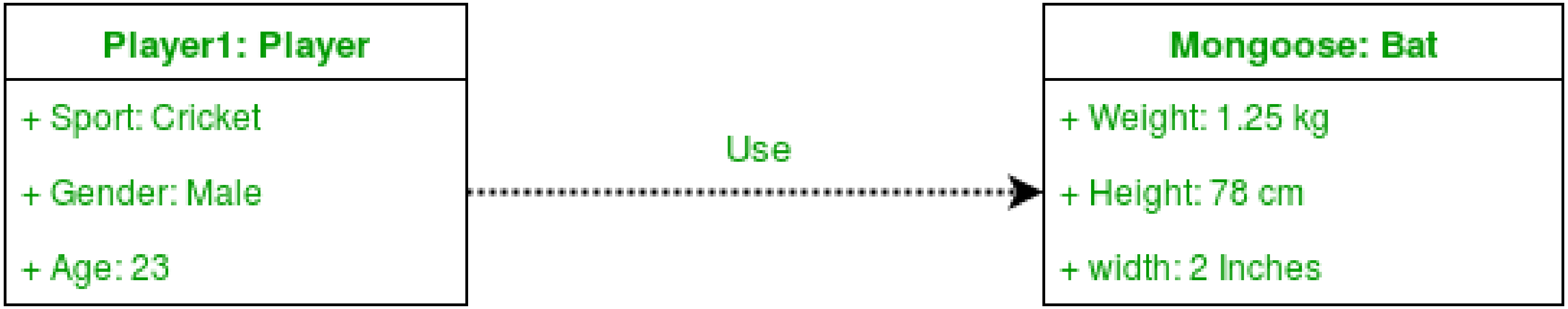
Key concepts: Dependency

- We use a **dependency** relationship to show when one element depends on another element.
- It points from the independent entity to the dependent entity in the system.
- This is a **unidirectional** kind of relationship between two objects.



Example: Dependency

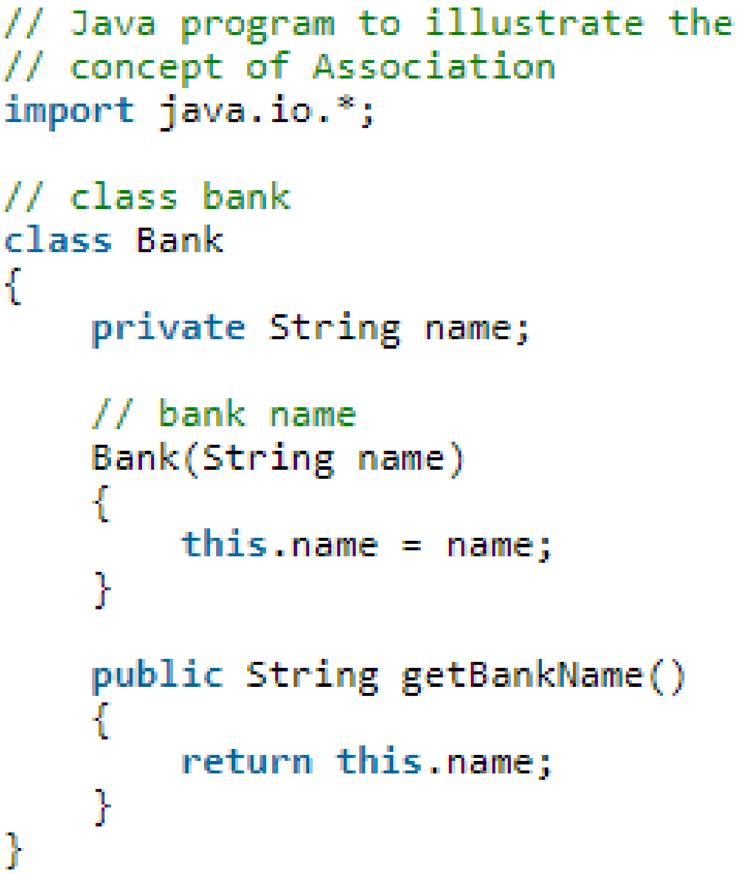
- In the figure below, an object of **Player** class is **dependent** (or **“uses”**) an object of **Bat** class.



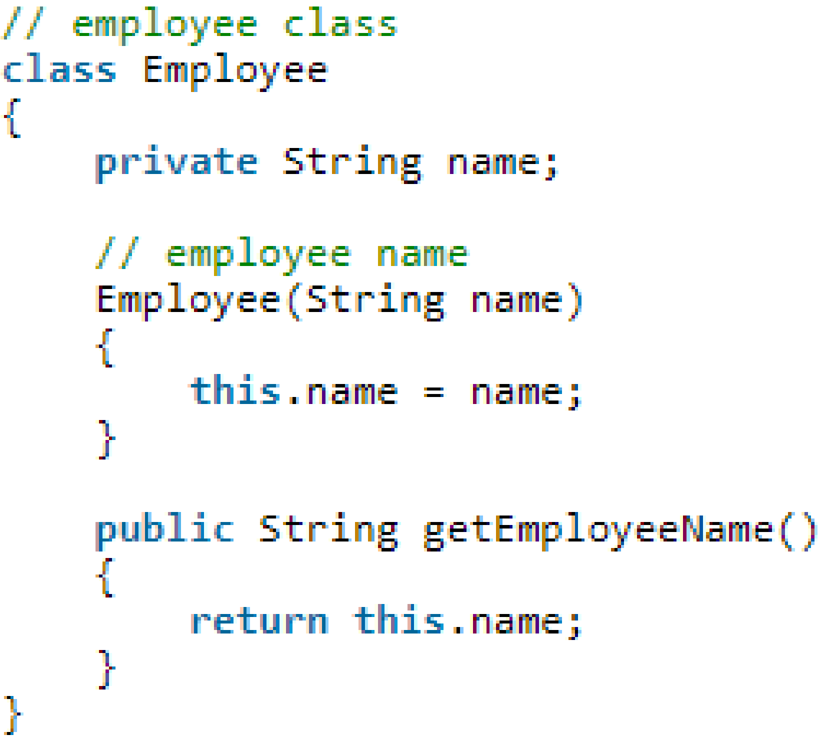
Key points: Association

- **Association** is relation between two separate classes which establishes through their Objects.
- **Association** can be one-to-one, one-to-many, many-to-one, many-to-many.
- In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object.
- **Aggregation** is a particular type of **Association**.

Example: Association



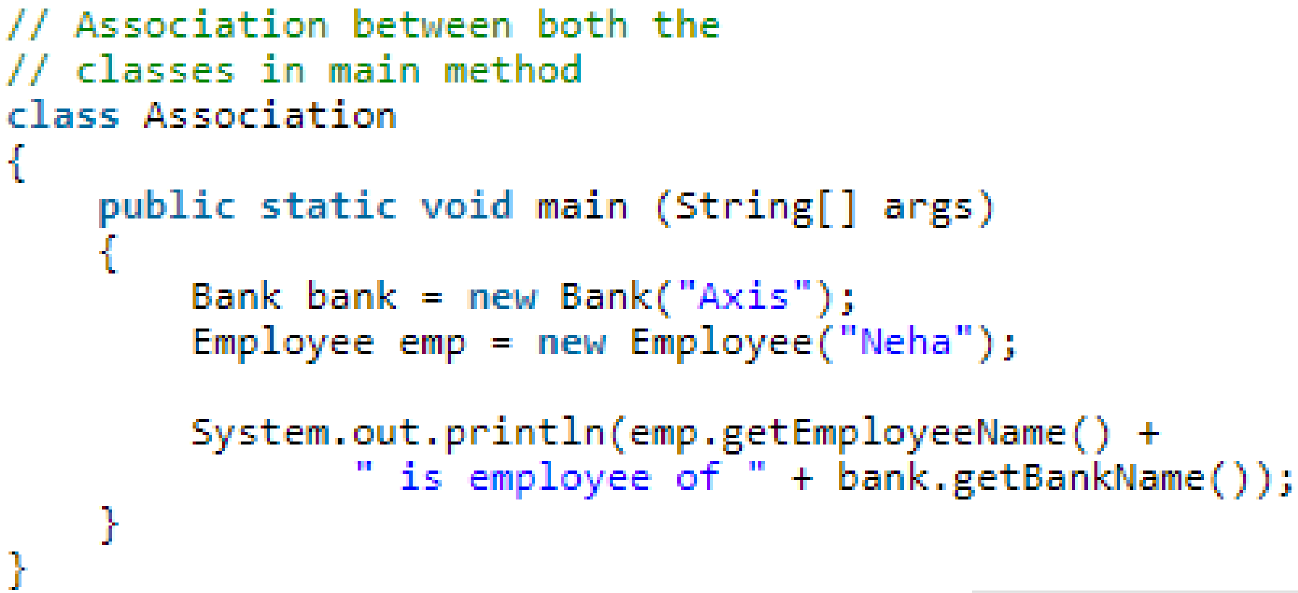
```java
// Java program to illustrate the
// concept of Association
import java.io.*;

// class bank
class Bank
{
    private String name;

    // bank name
    Bank(String name)
    {
        this.name = name;
    }

    public String getBankName()
    {
        return this.name;
    }
}
```



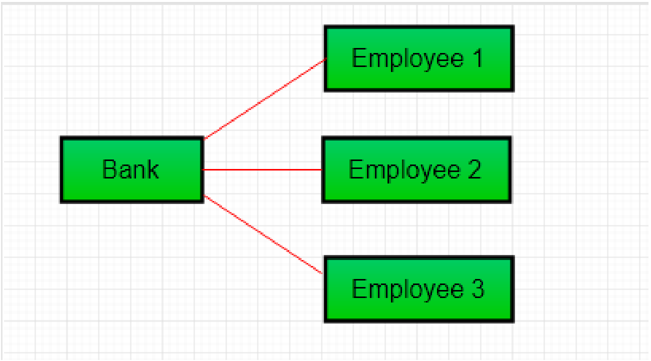
```java
// employee class
class Employee
{
    private String name;

    // employee name
    Employee(String name)
    {
        this.name = name;
    }

    public String getEmployeeName()
    {
        return this.name;
    }
}
```

Content developed by Dartford Grammar School Computer Science Department

Example: Association

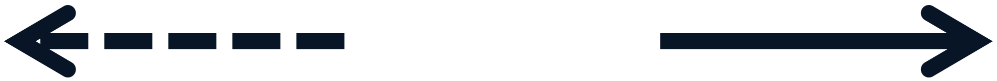


```
// Association between both the
// classes in main method
class Association
{
    public static void main (String[] args)
    {
        Bank bank = new Bank("Axis");
        Employee emp = new Employee("Neha");

        System.out.println(emp.getEmployeeName() +
               " is employee of " + bank.getBankName());
    }
}
```



Association vs Dependency

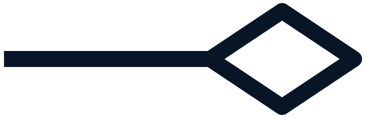
- Association and dependency are often confused in their usage.
- There are a large number of dependencies in a system.
- **We only represent the ones which are essential to convey for understanding the system**.
- We need to understand that every association implies a dependency itself.
- However, we prefer not to draw it separately.



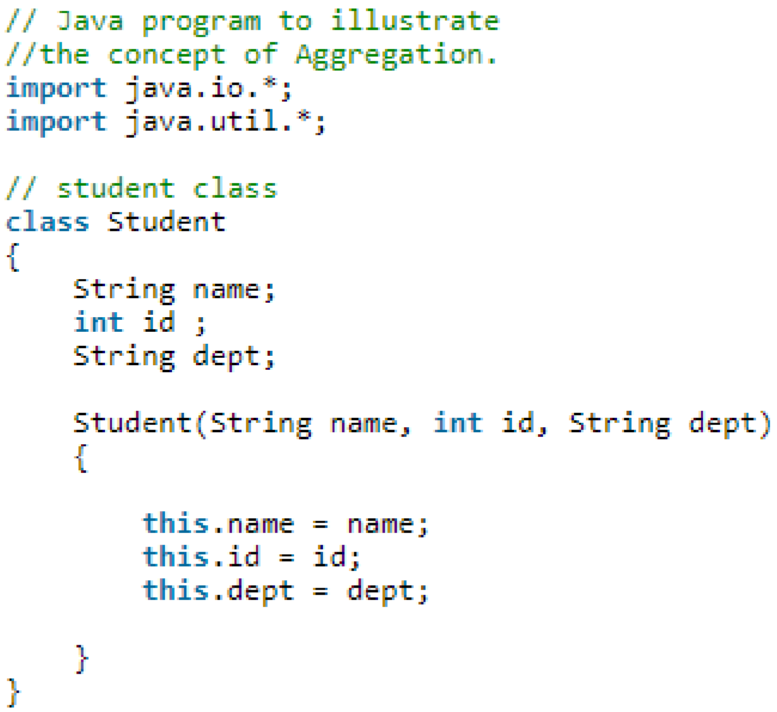
Key points: Aggregation

It is a special form of **Association** where:

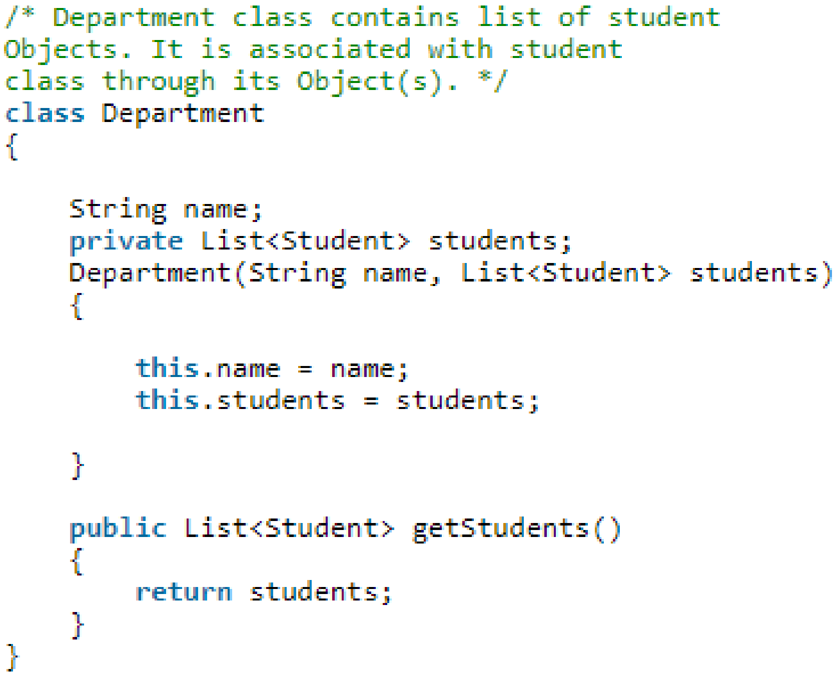
- It represents **“has a”** relationship.
- It is a **unidirectional association** i.e. a one way relationship. For example, department can have students but vice versa is not possible and thus unidirectional in nature.
- In **Aggregation**, **both the entries can survive individually** which means ending one entity will not effect the other entity



Example: Aggregation

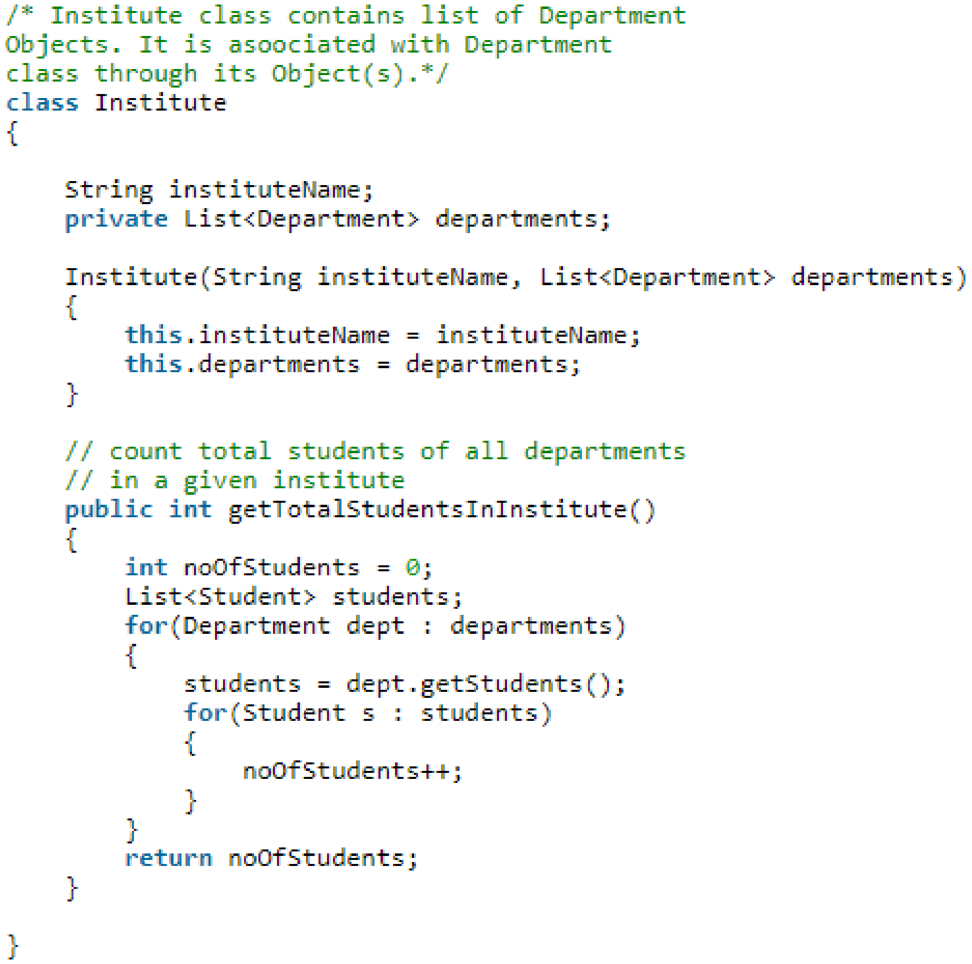


```
// Java program to illustrate
//the concept of Aggregation.
import java.io.*;
import java.util.*;

// student class
class Student
{
    String name;
    int id ;
    String dept;

    Student(String name, int id, String dept)
    {

        this.name = name;
        this.id = id;
        this.dept = dept;

    }
}
```



```
/* Department class contains list of student
Objects. It is associated with student
class through its Object(s). */
class Department
{

    String name;
    private List<Student> students;
    Department(String name, List<Student> students)
    {

        this.name = name;
        this.students = students;

    }

    public List<Student> getStudents()
    {
        return students;
    }
}
```

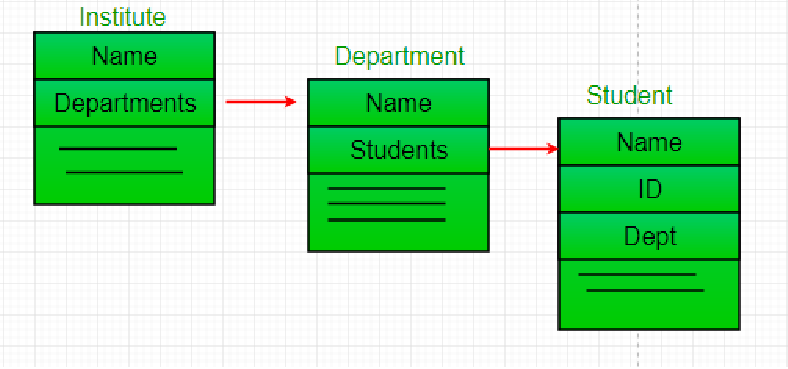
Example: Aggregation



```
/* Institute class contains list of Department
Objects. It is asoociated with Department
class through its Object(s).*/
class Institute
{

    String instituteName;
    private List<Department> departments;

    Institute(String instituteName, List<Department> departments)
    {
        this.instituteName = instituteName;
        this.departments = departments;
    }

    // count total students of all departments
    // in a given institute
    public int getTotalStudentsInInstitute()
    {
        int noOfStudents = 0;
        List<Student> students;
        for(Department dept : departments)
        {
            students = dept.getStudents();
            for(Student s : students)
            {
                noOfStudents++;
            }
        }
        return noOfStudents;
    }

}
```

Example: Aggregation

```
// main method
class GFG
{
    public static void main (String[] args)
    {
        Student s1 = new Student("Mia", 1, "CSE");
        Student s2 = new Student("Priya", 2, "CSE");
        Student s3 = new Student("John", 1, "EE");
        Student s4 = new Student("Rahul", 2, "EE");

        // making a List of
        // CSE Students.
        List <Student> cse_students = new ArrayList<Student>();
        cse_students.add(s1);
        cse_students.add(s2);

        // making a List of
        // EE Students
        List <Student> ee_students = new ArrayList<Student>();
        ee_students.add(s3);
        ee_students.add(s4);

        Department CSE = new Department("CSE", cse_students);
        Department EE = new Department("EE", ee_students);

        List <Department> departments = new ArrayList<Department>();
        departments.add(CSE);
        departments.add(EE);

        // creating an instance of Institute.
        Institute institute = new Institute("BITS", departments);

        System.out.print("Total students in institute: ");
        System.out.print(institute.getTotalStudentsInInstitute());
    }
}
```



Key points: Inheritance

- **Inheritance** is the mechanism by which one class is allow to inherit the features (states and behaviours) of another class.
- **Super Class:** The class whose features are inherited is known as super class (or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as sub class (or a derived class, extended class, or child class). The subclass can add its own states and behaviours in addition to the superclass states and behaviours.

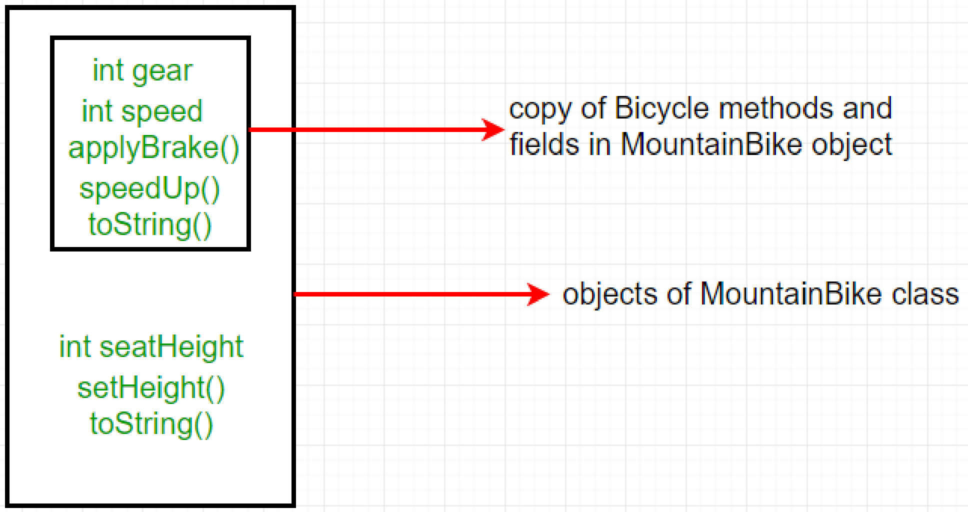
Example: Inheritance

```java
//Java program to illustrate the
// concept of inheritance

// base class
class Bicycle
{
    // the Bicycle class has two fields
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int gear, int speed)
    {
        this.gear = gear;
        this.speed = speed;
    }

    // the Bicycle class has three methods
    public void applyBrake(int decrement)
    {
        speed -= decrement;
    }

    public void speedUp(int increment)
    {
        speed += increment;
    }

    // toString() method to print info of Bicycle
    public String toString()
    {
        return("No of gears are "+gear
                +"\n"
                + "speed of bicycle is "+speed);
    }
}
```

```java
// derived class
class MountainBike extends Bicycle
{

    // the MountainBike subclass adds one more field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int gear,int speed,
                        int startHeight)
    {
        // invoking base-class(Bicycle) constructor
        super(gear, speed);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one more method
    public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }

    // overriding toString() method
    // of Bicycle to print more info
    @Override
    public String toString()
    {
        return (super.toString()+
                "\nseat height is "+seatHeight);
    }

}
```

Example: Inheritance

```java
// driver class
public class Test
{
    public static void main(String args[])
    {

        MountainBike mb = new MountainBike(3, 100, 25);
        System.out.println(mb.toString());

    }
}
```



int gear
int speed
applyBrake()
speedUp()
toString()

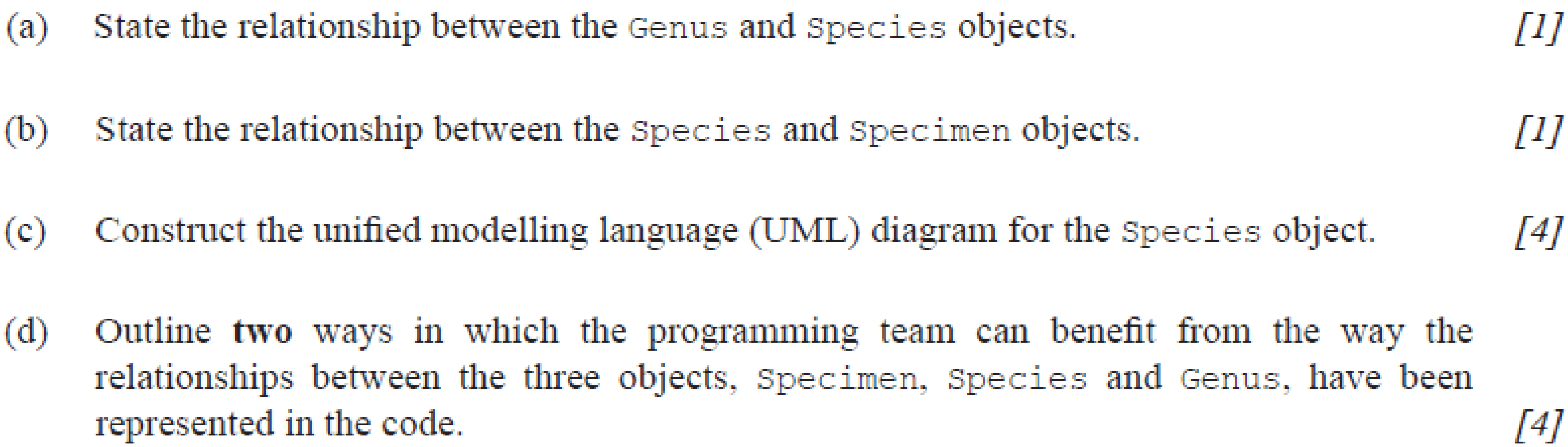
int seatHeight
setHeight()
toString()

copy of Bicycle methods and fields in MountainBike object

objects of MountainBike class

Content developed by Dartford Grammar School Computer Science Department

Exam style question:



(a) State the relationship between the `Genus` and `Species` objects. *[1]*

(b) State the relationship between the `Species` and `Specimen` objects. *[1]*

(c) Construct the unified modelling language (UML) diagram for the `Species` object. *[4]*

(d) Outline **two** ways in which the programming team can benefit from the way the relationships between the three objects, `Specimen`, `Species` and `Genus`, have been represented in the code. *[4]*

Important to note:

- ✓ Know **how to identify** relationships both in **UML** and **Java**.
- ✓ The two big ones is **inheritance** (“*is a*”) and **dependence** (“*uses a*”)
- ✓ Know **WHY** we use these relationships