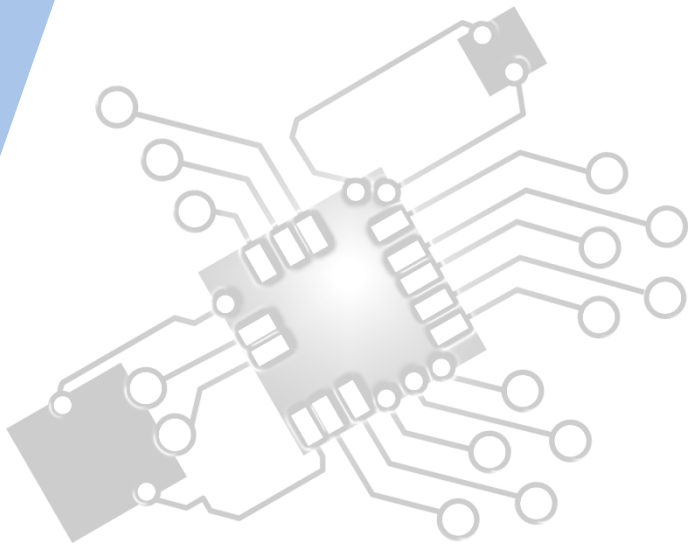
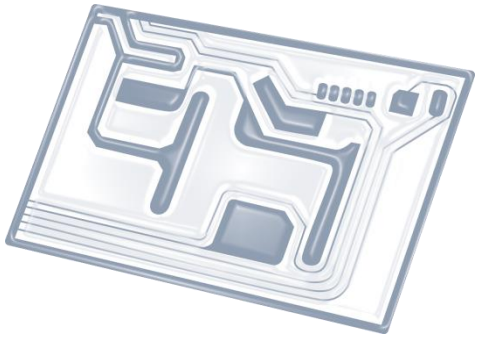
Objects as a programming concept

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

HL & SL D.3 Overview

D.3 Program development

D.3.1 Define the terms: class, identifier, primitive, instance variable, parameter variable, local variable

D.3.2 Define the terms: method, accessor, mutator, constructor, signature, return value

D.3.3 Define the terms: private, protected, public, extends, static

D.3.4 Describe the uses of the primitive data types and the reference class string

D.3.5 Construct code to implement assessment statements

D.3.6 Construct code examples related to selection statements

D.3.7 Construct code examples related to repetition statements

D.3.8 Construct code examples related to static arrays

D.3.9 Discuss the features of modern programming languages that enable internationalization

D.3.10 Discuss the ethical and moral obligations of programmers

1: System design

2: Computer Organisation

3: Networks

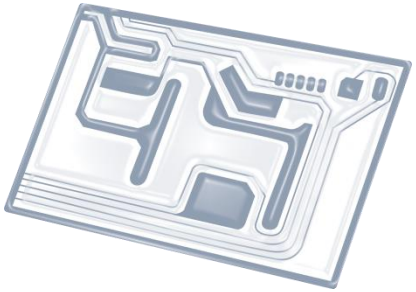
4: Computational thinking

5: Abstract data structures

6: Resource management

7: Control

D: OOP



Topic D.3.3

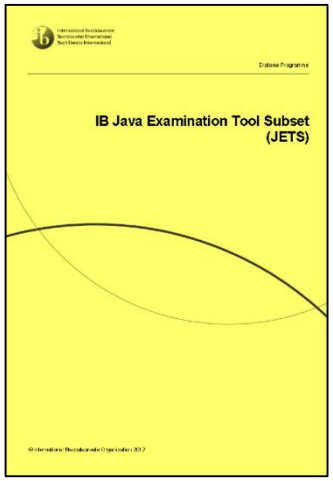
Define the terms: **private**, **protected**, **public**, **extends**, **static**

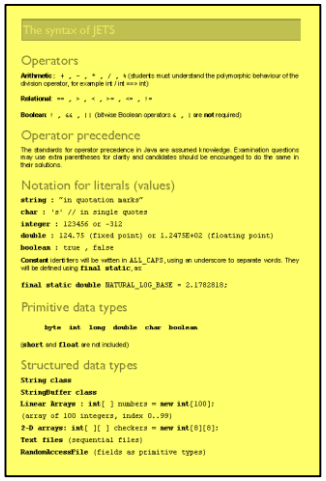
**"My computer freezes up 3 times a day!
That's why I don't believe in global warming."**

Exam note!

This curriculum point relates closely to the details published in the **JETS booklet**.

You will **NOT get a copy** of this booklet in the Paper 2 exam.





Access modifiers

- Access level modifiers determine whether other classes can use a particular field or invoke a particular method.
- A class may be declared with the modifier **`public`**, in which case that class is **visible to all classes everywhere**.
- If a class has no modifier (i.e. the default position) it is visible only within its own package.
- There are three Java access modifiers:
  - **`public`**
  - **`protected`**
  - **`private`**

Definition: public

- A class, method, field or constructor that is declared **`public`** can be **accessed from any other class**.
- Therefore, fields, methods, blocks declared inside a **`public`** class can be **accessed from any class** belonging to the Java Universe.
- Because of class inheritance, all **`public`** methods and variables of a class are inherited by its subclasses.

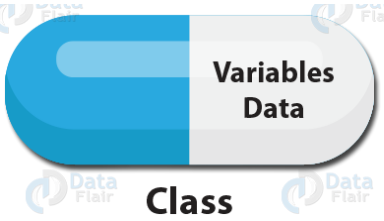
Example: public

```
public class MathUtil {

    public double cubedRoot(int num) {
        return Math.pow(num, 1.0/3);
    }

    public double circumference(double radius) {
        return 2*Math.PI*radius;
    }

    public double area(double radius) {
        return Math.PI *radius*radius;
    }

}
```

These routines and variables can be **accessed from anywhere**.
This means if you have a **MathUtil** object in any other class, you can use its public methods and variables

```
public class MyProgram {

    public static void main(String[] args) {
        MathUtil math = new MathUtil();

        //area of circle with radius 3
        double circleArea = math.area(3);

        //cubed root of 64
        float root = (float) math.cubedRoot(64);

        System.out.println("Area of circle radius 3: " + circleArea);
        System.out.println("Cubed root of 64: " + root);
    }

}
```

See D.2.1



Definition: private

- Methods, variables, and constructors that are declared **`private`** can only be accessed **within the declared class itself**.
- **`private`** is the most restrictive access level.
- **Classes cannot be `private`**, but methods and variables can.
- Variables that are declared **`private`** can be accessed outside the class, if **`public`** getter methods are present in the class.
- Using the **`private`** modifier is the main way that an object **encapsulates** itself and **hides data** from the outside world.

Example: private

```
public class Dog extends Animal {

    private int numberOfLegs;
    private boolean hasOwner;

    public Dog() {
        numberOfLegs = 4;
        hasOwner = false;
    }

    private void bark() {
        System.out.println("Woof!");
    }

    public void move() {
        System.out.println("Running");
    }

}
```

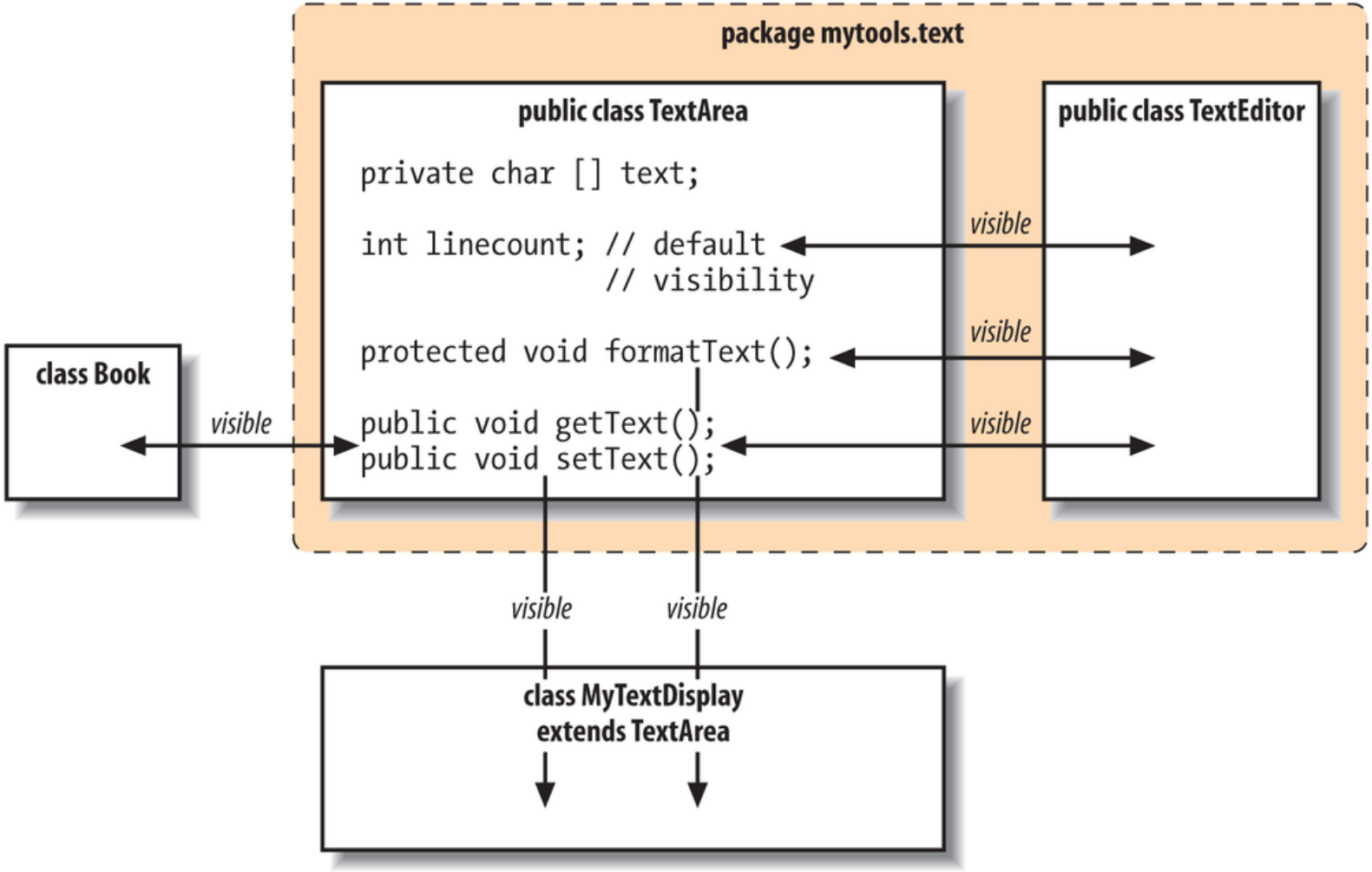
```
public class Cat extends Animal {

    public void makeDogBark() {
        Dog d = new Dog();
        d.bark();
    }

    public void meow() {
        System.out.println("Meow!");
    }

    public void move() {
        System.out.println("Prancing");
    }

}
```

```
public class Dog extends Animal {

    private int numberOfLegs;
    private boolean hasOwner;

    public Dog() {
        numberOfLegs = 4;
        hasOwner = false;
    }

    public void makeDogBark() {
        Dog d = new Dog();
        d.bark();
    }

    private void bark() {
        System.out.println("Woof!");
    }

    public void move() {
        System.out.println("Running");
    }

}
```

Definition: protected

- Variables, methods, and constructors, which are declared **`protected`** in a superclass can be **accessed only by the subclasses** in any class within the package of the protected members' class.
- **`protected`** cannot be applied to classes.
- **`protected`** access gives the **subclass** a chance to use the helper method or variable, while **preventing** a **nonrelated** class from trying to use it.

Example: protected



Example 2: protected

```
public abstract class Animal {

    public boolean isAPet = true;
    public String owner = "Fred";

    public void sleep() {
        System.out.println("Sleeping");
    }

    protected void eat() {
        System.out.println("Eating");
    }

    public abstract void move();

}
```

```
public class Dog extends Animal {

    private int numberOfLegs;
    private boolean hasOwner;

    public Dog() {
        numberOfLegs = 4;
        hasOwner = false;
    }

    public void makeDogBark() {
        Dog d = new Dog();
        d.bark();
    }

    public void makeDogEat() {
        eat();
    }

    private void bark() {
        System.out.println("Woof!");
    }

    public void move() {
        System.out.println("Running");
    }

}
public class Chair {

    public void makeAnimalEat() {
        Dog d = new Dog();
        d.eat();
    }

}
```

See D.2.2

Definition: extends

`extends` is the keyword used in a sub class to inherit the properties of a super class

```
class Super {
    .....
    .....
}
class Sub extends Super {
    .....
    .....
}
```

Example: extends

```
class Calculation {
   int z;

   public void addition(int x, int y) {
      z = x + y;
      System.out.println("The sum of the given numbers:"+z);
   }

   public void Subtraction(int x, int y) {
      z = x - y;
      System.out.println("The difference between the given numbers:"+z);
   }
}

public class My_Calculation extends Calculation {
   public void multiplication(int x, int y) {
      z = x * y;
      System.out.println("The product of the given numbers:"+z);
   }

   public static void main(String args[]) {
      int a = 20, b = 10;
      My_Calculation demo = new My_Calculation();
      demo.addition(a, b);
      demo.Subtraction(a, b);
      demo.multiplication(a, b);
   }
}
```

Definition: static

`static` is a non-access modifier in Java which is applicable for the following:

- blocks
- variables
- methods
- nested classes

To create a static member (variable, method, etc.), precede its declaration with the keyword **`static`**.

`static` variables are used for any property that is common to all objects of that type.

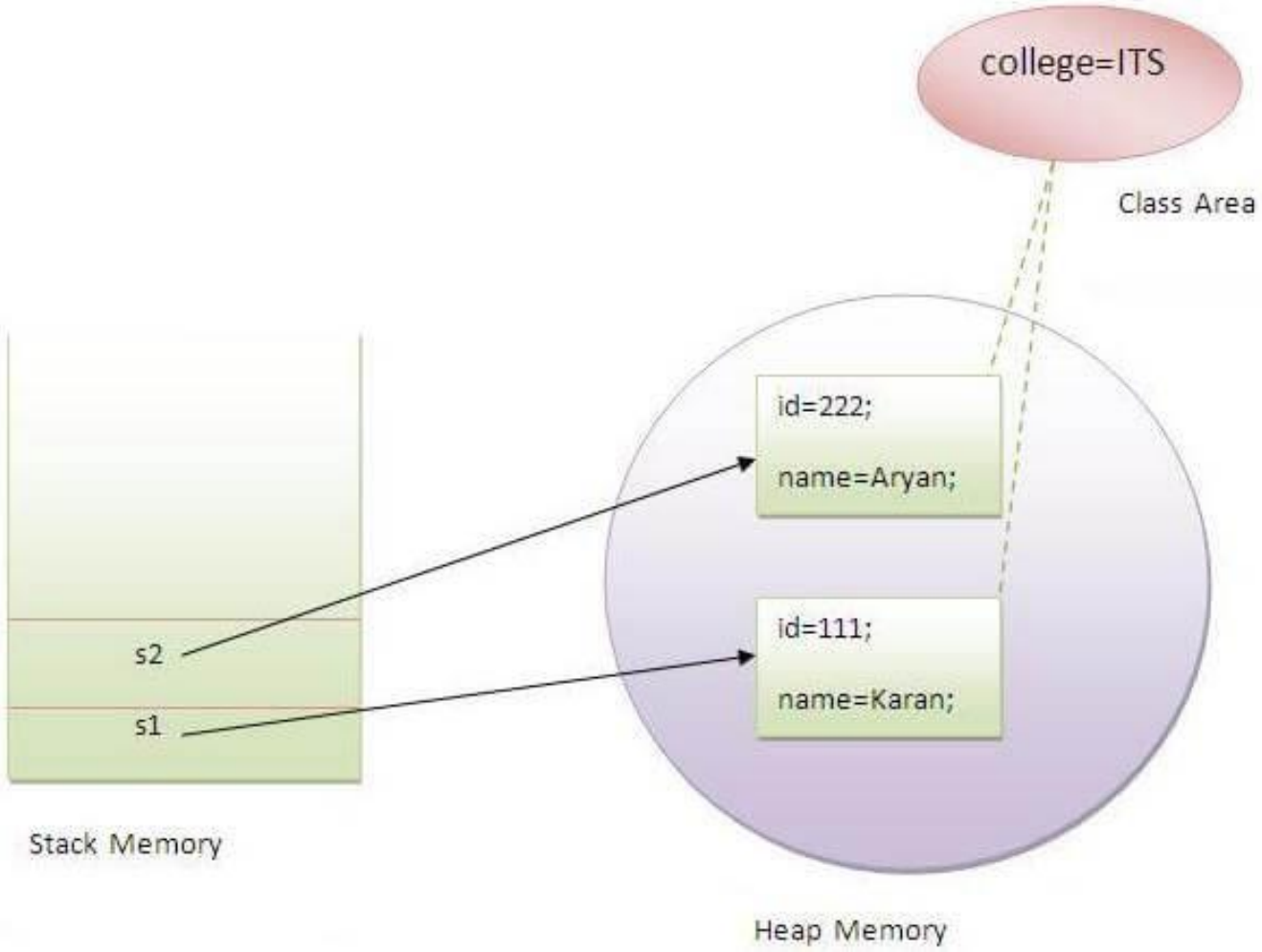
Example 1: static

```
class Student{
    int rollno;
    String name;
    String college="ITS";
}
```

```
class Student8{
  int rollno;
  String name;
  static String college ="ITS";

  Student8(int r,String n){
  rollno = r;
  name = n;
  }
 void display (){System.out.println(rollno+" "+name+" "+college);}

 public static void main(String args[]){
 Student8 s1 = new Student8(111,"Karan");
 Student8 s2 = new Student8(222,"Aryan");

 s1.display();
 s2.display();
 }
}
```

```
Output:111 Karan ITS
       222 Aryan ITS
```

Example 2: static



More on static

- Declares a static member of a class that will be the same for all members.
- The **static** keyword in Java means that the variable or function is **shared between all instances** of that class as it belongs to the type, not the actual objects themselves.
- So if you have a variable: private **static** int i = 0; and you increment it (i++) in one instance, the change will be reflected in all instances.

Also see: http://www.javatpoint.com/static-keyword-in-java

Summary of access modifiers

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
`no modifier`	Y	Y	N	N
`private`	Y	N	N	N

- The first data column indicates whether the class itself has access to the member defined by the access level - **a class always has access to its own members**.
- The second column indicates whether classes in the same package as the class have access to the member.
- The third column indicates whether subclasses of the class declared outside this package have access to the member.
- The fourth column indicates whether all classes have access to the member.