



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.3.1

Define the terms: **class**, **identifier**,
primitive, **instance variable**,
parameter variable, **local variable**

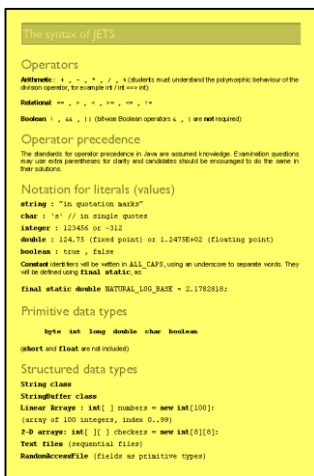
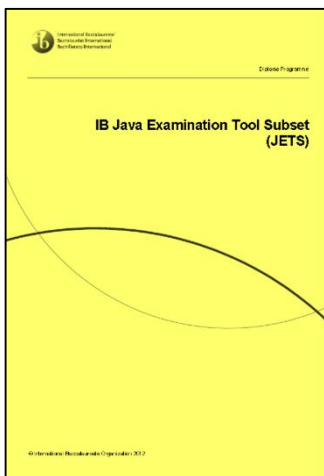




Exam note!

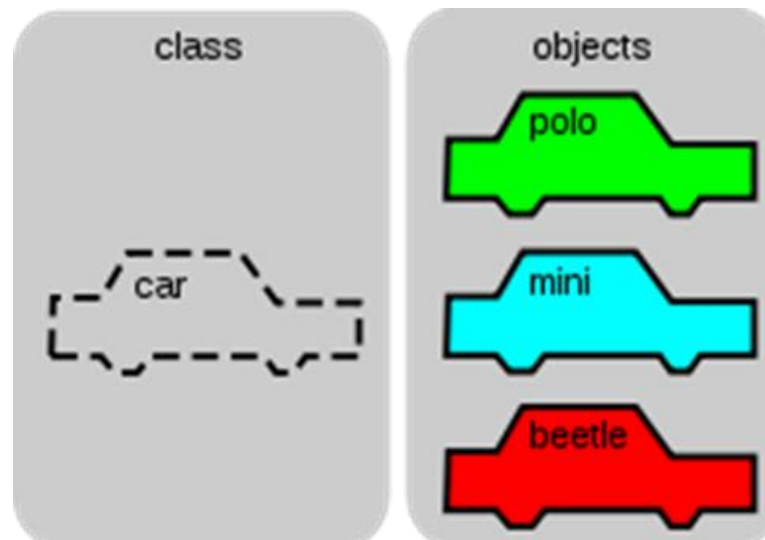
This curriculum point relates closely to the details published in the JETS booklet.

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



Definition: **class**

Class – an extensible **program-code-template** for creating objects, providing **initial values** for states (variables) and implementations of behaviours (functions/procedures/methods)



Example: class

```
public class Animal {
```

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

States/Data/Fields

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

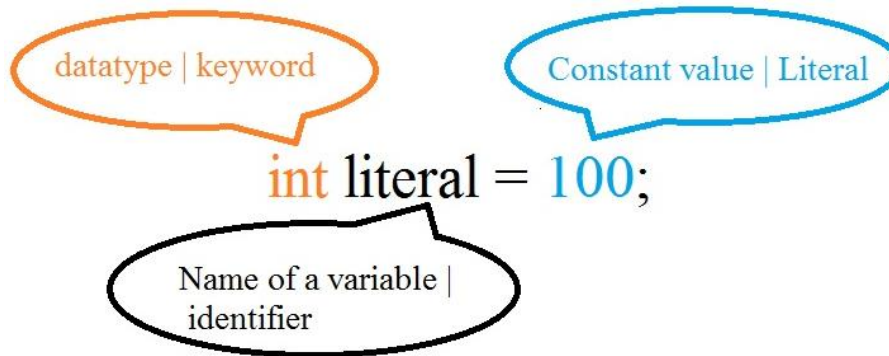
```
    public void eat() {  
        System.out.println("Eating");  
    }
```

**Behaviours/
Actions/
Methods**

```
}
```

Definition: **identifier**

- An identifier is a **named pointer** that **explicitly identifies** an object, class, method, or variable.
- It allows a programmer to refer to the item from other places in the program.
- To make the most out of the identifiers you choose make them meaningful and follow the standard Java **naming conventions**.



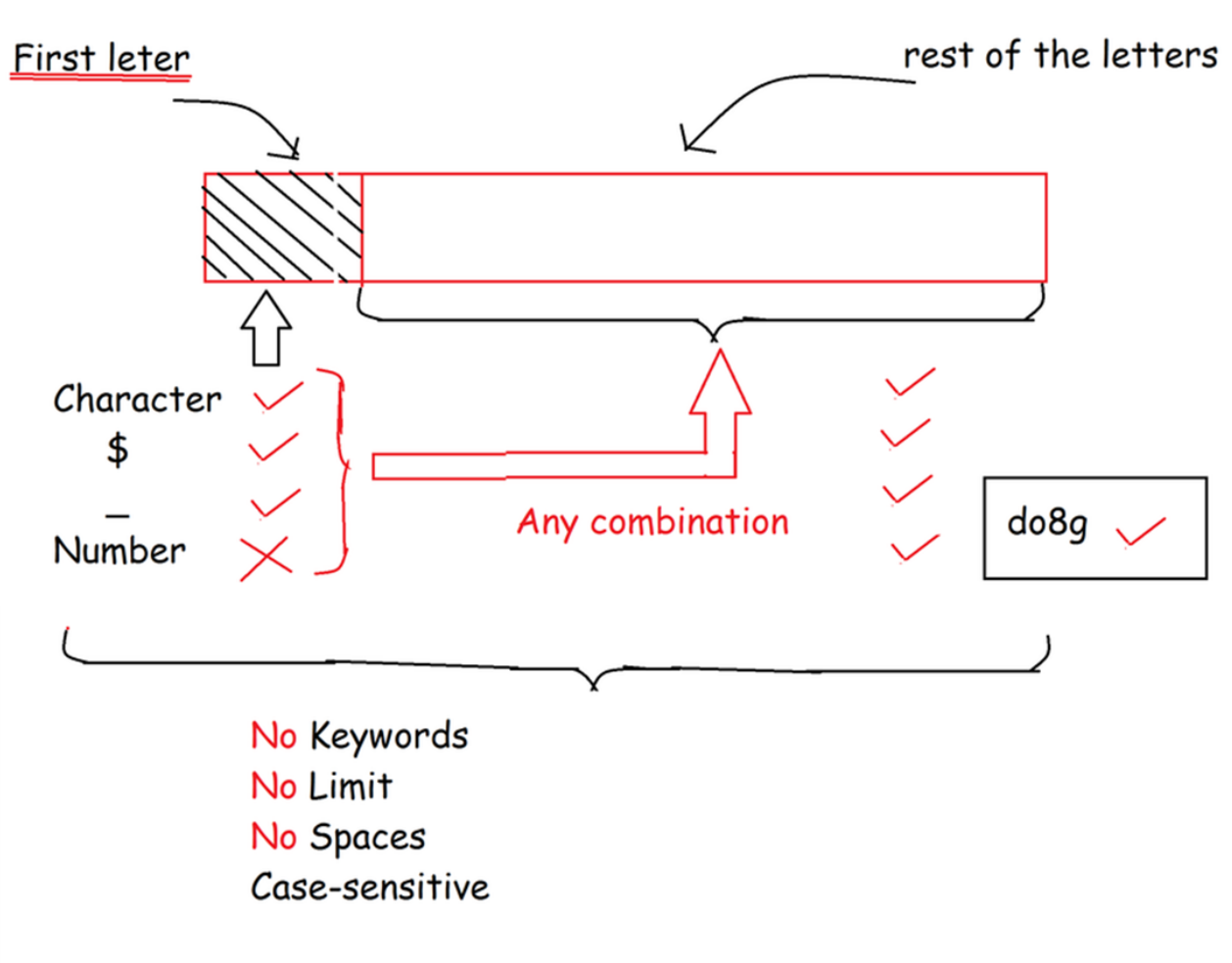
Example: identifier

```
public readStudentName()  
{  
    System.out.println("Enter student name");  
    String answer = kb.nextLine();  
}
```

More info: rules for identifiers (Java)

Example :

dog ✓
\$dog ✓
_dog ✓
8dog ✗



Definition: **primitive**

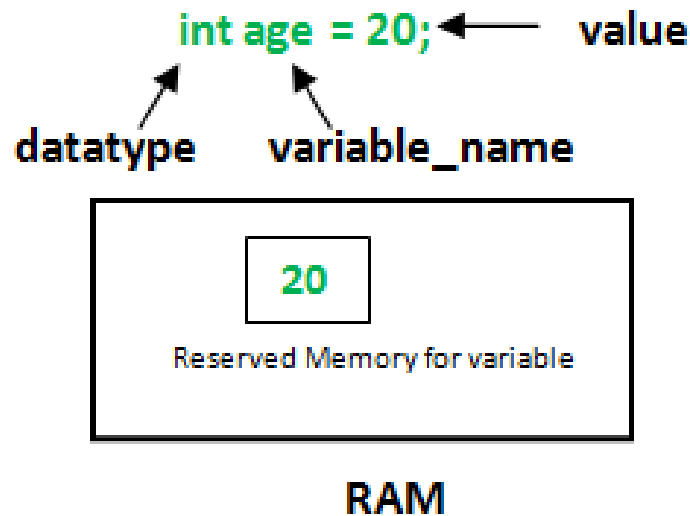
- Primitive types are the **most basic data types** available within the Java language.
- There are 8: **boolean, byte, char, short, int, long, float and double.**
- These types serve as the building blocks of data manipulation in Java.
- Such types serve only one purpose: **to contain pure, simple values of a particular kind.**

Example: primitives

Primitive Type	Size	Minimum Value	Maximum Value
char	16-bit	Unicode 0	Unicode $2^{16}-1$
byte	8-bit	-128	+127
short	16-bit	-2^{15} (-32,768)	$+2^{15}-1$ (32,767)
int	32-bit	-2^{31} (-2,147,483,648)	$+2^{31}-1$ (2,147,483,647)
long	64-bit	-2^{63} (-9,223,372,036,854,775,808)	$+2^{63}-1$ (9,223,372,036,854,775,807)
float	32-bit	32-bit floating-point numbers	
double	64-bit	64-bit floating-point numbers	
boolean	1-bit	true or false	

Definition: **variable**

- A variable provides us with **named storage location** for a value that a program can manipulate.
- They must be declared before they can be used and can only **contain data of a particular type** (in Java).



Definition: **instance variable**

- Instance variables are non-static variables and are declared **in a class outside any method**, constructor or block.
- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access modifiers (public/private/protected) for instance variables.
- If we do not specify any access modifier then the default access modifier of the class will be used.

Example: instance variables

```
import java.io.*;
class Marks
{
    //These variables are instance variables.
    //These variables are in a class and are not inside any function
    int engMarks;
    int mathsMarks;
    int phyMarks;
}
```

} instance variables

```
class MarksDemo
{
    public static void main(String args[])
    {
        //first object
        Marks obj1 = new Marks();
        obj1.engMarks = 50;
        obj1.mathsMarks = 80;
        obj1.phyMarks = 90;

        //second object
        Marks obj2 = new Marks();
        obj2.engMarks = 80;
        obj2.mathsMarks = 60;
        obj2.phyMarks = 85;

        //displaying marks for first object
        System.out.println("Marks for first object:");
        System.out.println(obj1.engMarks);
        System.out.println(obj1.mathsMarks);
        System.out.println(obj1.phyMarks);

        //displaying marks for second object
        System.out.println("Marks for second object:");
        System.out.println(obj2.engMarks);
        System.out.println(obj2.mathsMarks);
        System.out.println(obj2.phyMarks);
    }
}
```

Output:

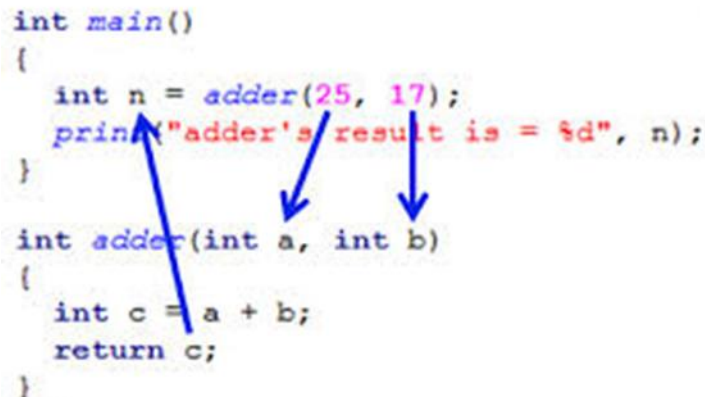
```
Marks for first object:
50
80
90
Marks for second object:
80
60
85
```

Definition: **parameter variable**

- **Parameters** allow us to **pass information** or instructions into functions and procedures.
- **Parameters** are the names of the information that we want to use in a function or procedure.
- The **values** passed in are called **arguments**.

```
int main()
{
    int n = adder(25, 17);
    printf("adder's result is = %d", n);
}

int adder(int a, int b)
{
    int c = a + b;
    return c;
}
```



Example 1: parameter variable

```
void go() {  
    TestStuff t = new TestStuff();  
    t.takeTwo(12, 34);  
}
```

```
void takeTwo(int x, int y) {  
    int z = x + y;  
    System.out.println("Total is " + z);  
}
```

The arguments you pass land in the same order you passed them. First argument lands in the first parameter, second argument in the second parameter, and so on.

Example 2: parameter variable

//Invoke (call) the method

```
int number1 = 25;  
int number2 = 47;  
int sum = add(number1, number2);
```

actual parameters
(or arguments)

//Method definition

```
public int add(int x, int y)  
{  
    return (x + y);  
}
```

formal parameters

Definition: **local variable**

- A variable defined **within a block or method or constructor** is called local variable.
- These variable are created when the block is entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The **scope** of these variables exists only within the block in which the variable is declared; we can **access these variables only within that block**.

Example: local variable

```
public class StudentDetails
{
    public void StudentAge()
    {
        //local variable age
        int age = 0;
        age = age + 5;
        System.out.println("Student age is : " + age);
    }

    public static void main(String args[])
    {
        StudentDetails obj = new StudentDetails();
        obj.StudentAge();
    }
}
```

Output:

```
Student age is : 5
```