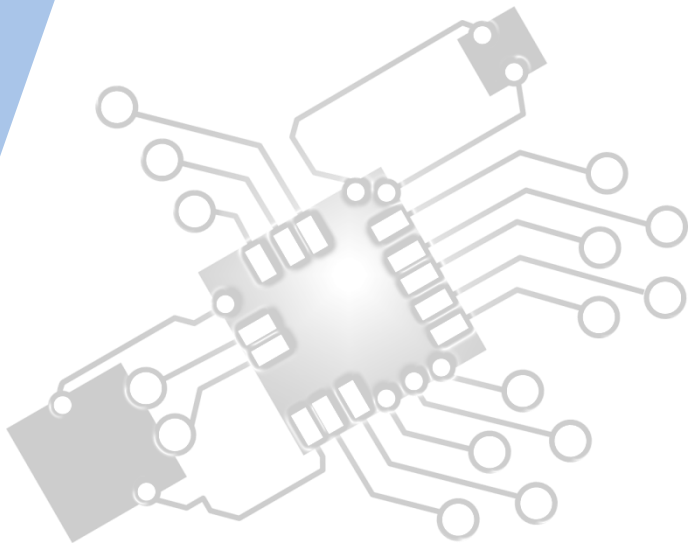
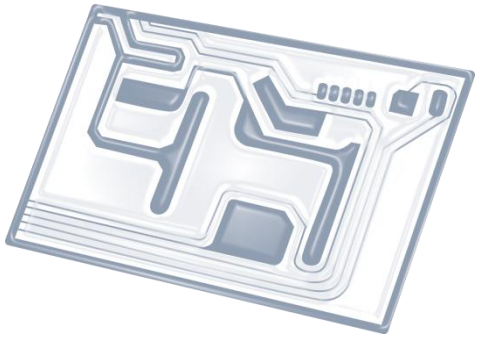
Objects as a programming concept

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.2 Overview

D.2 Features of OOP

D.2.1 Define the term encapsulation

D.2.2 Define the term inheritance

D.2.3 Define the term polymorphism

D.2.4 Explain the advantages of encapsulation

D.2.5 Explain the advantages of inheritance

D.2.6 Explain the advantages of polymorphism

D.2.7 Describe the advantages of libraries of objects

D.2.8 Describe the disadvantages of OOP

D.2.9 Discuss the use of programming teams

D.2.10 Explain the advantages of modularity in program development

1: System design

2: Computer Organisation

3: Networks

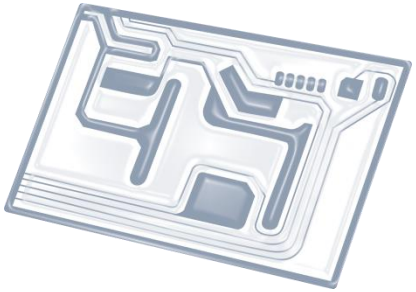
4: Computational thinking

5: Abstract data structures

6: Resource management

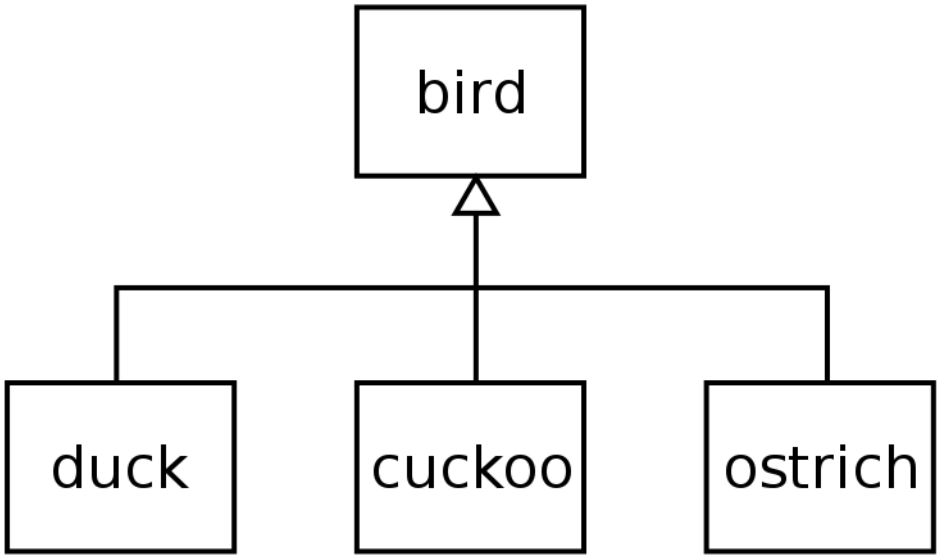
7: Control

D: OOP



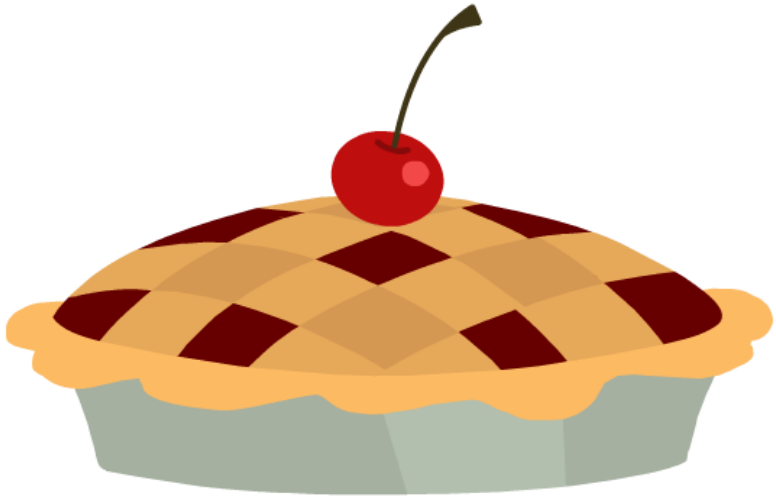
Topic D.2.2

Define the term: **inheritance**



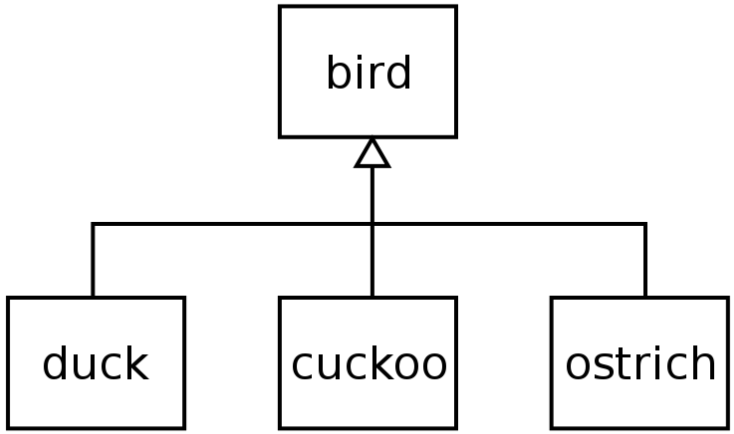
Four OOP fundamentals:

- **A**bstraction
- **P**olymorphism
- **I**nheritance
- **E**ncapsulation

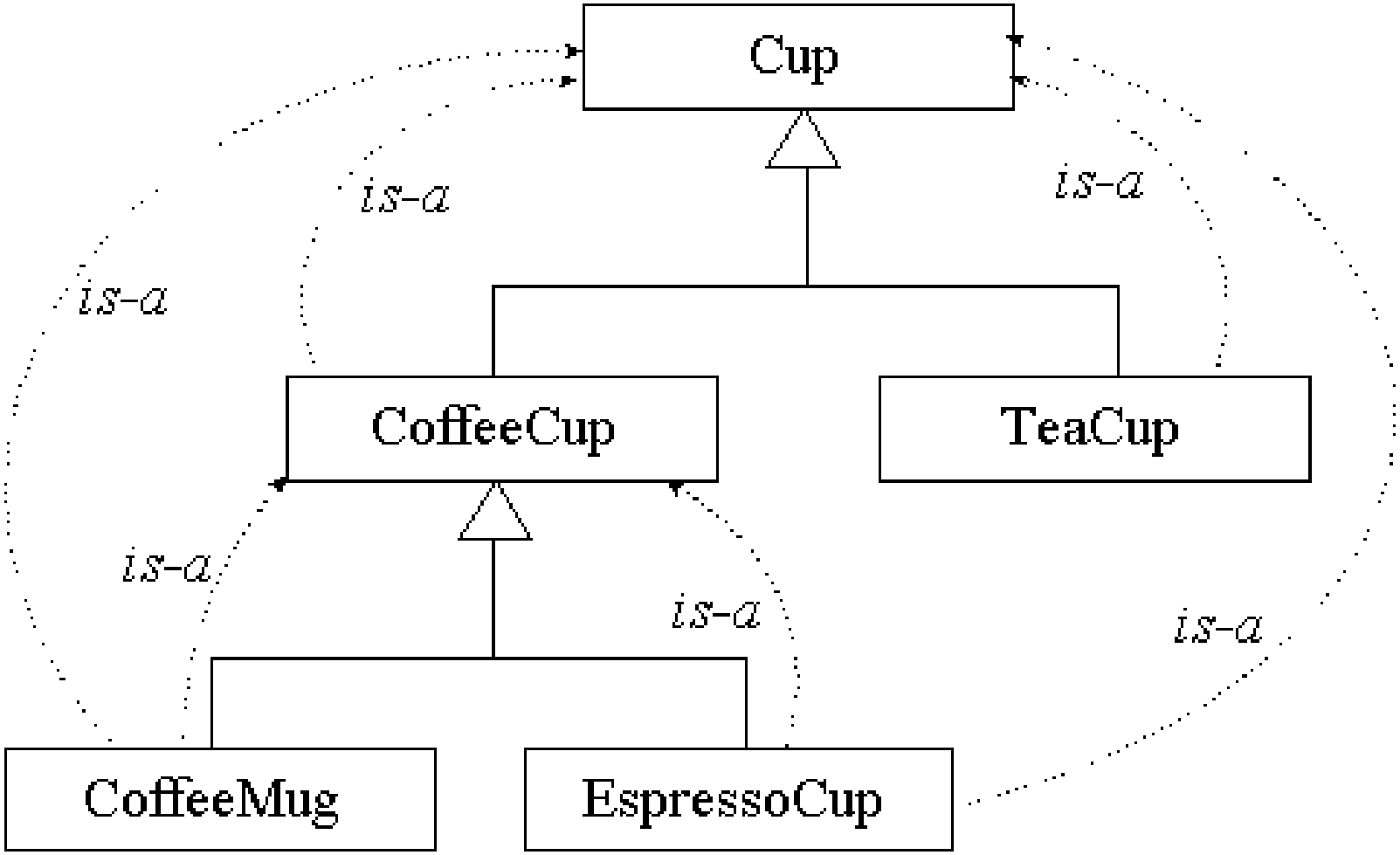


Definition: Inheritance

- Process whereby one object acquires the **properties** (states and behaviours) of another
- The most commonly used keyword would be **extends** and **implements**



Example


Cup
is-a
is-a
is-a
CoffeeCup
TeaCup
is-a
is-a
is-a
CoffeeMug
EspressoCup

Coding example:

```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

Now, based on the previous example, in OOP terms, the following are true:

- **Animal** is the superclass of **Mammal** class.
- **Animal** is the superclass of **Reptile** class.
- **Mammal** and **Reptile** are subclasses of **Animal** class.
- **Dog** is the subclass of both **Mammal** and **Animal** classes.

Now, if we consider the **IS-A** relationship, we can say:

- Mammal **IS-A** Animal
- Reptile **IS-A** Animal
- Dog **IS-A** Mammal
- *Hence*: Dog **IS-A** Animal as well

With use of the **extends** keyword the subclasses will be able to inherit all the properties of the superclass **except for the private** properties of the superclass.

Example

```
public class Dog extends Mammal{

   public static void main(String args[]){

      Animal a = new Animal();
      Mammal m = new Mammal();
      Dog d = new Dog();

      System.out.println(m instanceof Animal);
      System.out.println(d instanceof Mammal);
      System.out.println(d instanceof Animal);
   }
}
```

This would produce the following result:

```
true
true
true
```

Example 2

Animal.java:

```
01 public class Animal {
02     public Animal() {
03         System.out.println("A new animal has been created!");
04     }
05
06     public void sleep() {
07         System.out.println("An animal sleeps...");
08     }
09
10     public void eat() {
11         System.out.println("An animal eats...");
12     }
13 }
```

A type of animal

Bird.java:

```
01 public class Bird extends Animal {
02     public Bird() {
03         super();
04         System.out.println("A new bird has been created!");
05     }
06 
07     @Override
08     public void sleep() {
09         System.out.println("A bird sleeps...");
10     }
11 
12     @Override
13     public void eat() {
14         System.out.println("A bird eats...");
15     }
16 }
```

Another type of animal

Dog.java:

```
01 public class Dog extends Animal {
02     public Dog() {
03         super();
04         System.out.println("A new dog has been created!");
05     }
06 
07     @Override
08     public void sleep() {
09         System.out.println("A dog sleeps...");
10     }
11 
12     @Override
13     public void eat() {
14         System.out.println("A dog eats...");
15     }
16 }
```

The Driver class

MainClass.java:

```
01 public class MainClass {
02     public static void main(String[] args) {
03         Animal animal = new Animal();
04         Bird bird = new Bird();
05         Dog dog = new Dog();
06 
07         System.out.println();
08 
09         animal.sleep();
10         animal.eat();
11 
12         bird.sleep();
13         bird.eat();
14 
15         dog.sleep();
16         dog.eat();
17     }
18 }
```

Output

```
A new animal has been created!
A new animal has been created!
A new bird has been created!
A new animal has been created!
A new dog has been created!

An animal sleeps...
An animal eats...
A bird sleeps...
A bird eats...
A dog sleeps...
A dog eats...
```