Introduction

OOP1 Review

COMP3607
Object Oriented Programming II

04-Sept-2019

Outline

- Course overview
- Review
  - OOP1 Concepts
  - Object-Oriented Design Principles

Review

- Classes vs Objects
- Instantiation, Initialisation
- Encapsulation and Abstraction
- Associations
- Inheritance
- Polymorphism

Classes vs Objects

A class defines a set of attributes and behaviour.

An object is a concrete manifestation of a class.

Class	Object
1) Class is a collection of similar objects	1) Object is an instance of a class
2) Class is conceptual (is a template)	2) Object is real
3) No memory is allocated for a class.	3) Each object has its own memory
4) Class can exist without any objects	4) Objects can't exist without a class
5) Class does not have any values associated with the fields	5) Every object has its own values associated with the fields

Exercise

Identify the classes

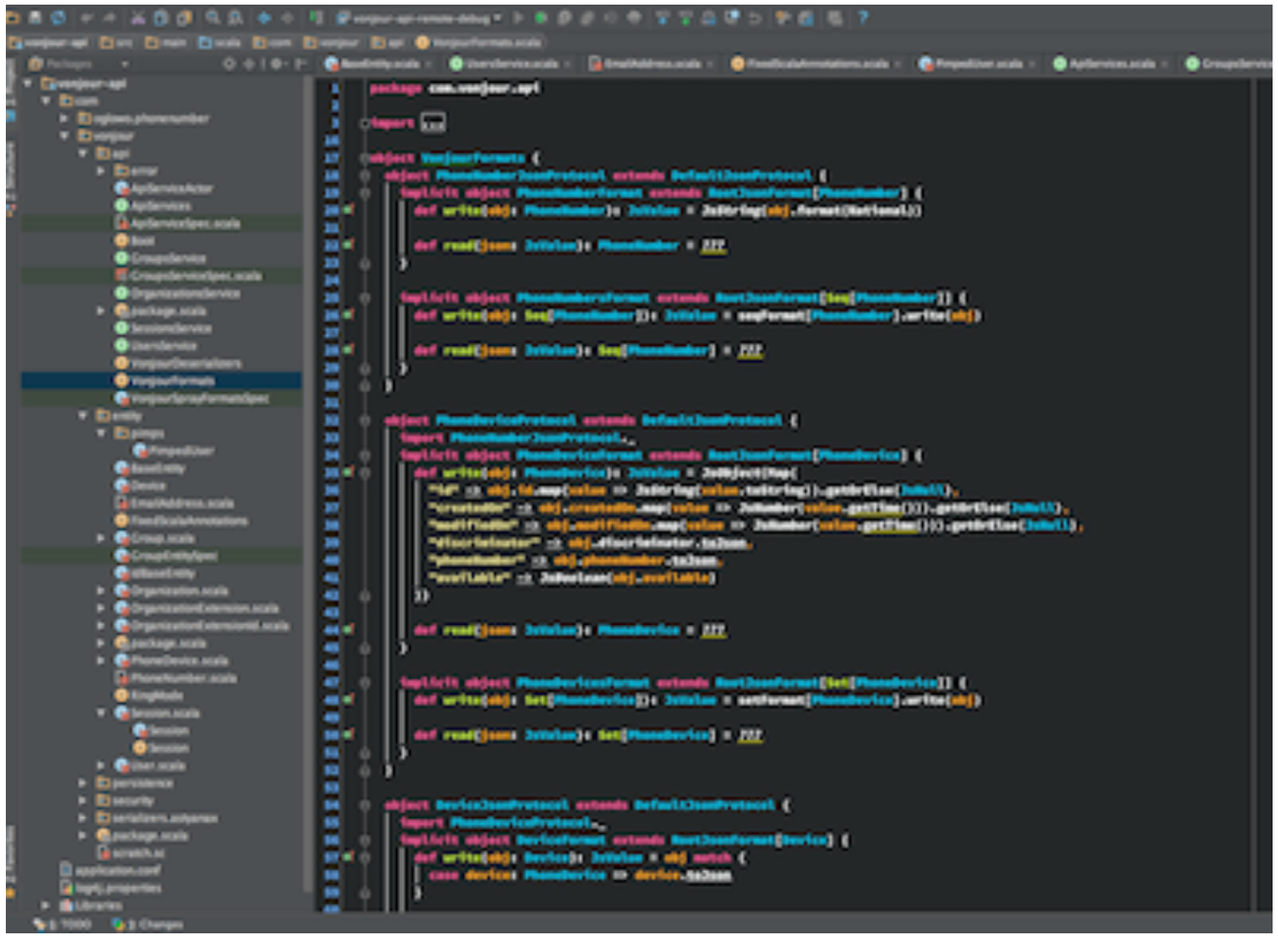
```
String s;
int sum = 1000;
ArrayList list = new ArrayList();
```

Classes vs Objects

Solution

```
String s;
int sum = 1000;
ArrayList list = new ArrayList();
```

IDEs can help us out with colour coded classes



Exercise

Identify the objects and classes

```
String s;
double delta = 1.89;
boolean equal = false;
Object object;
Integer count = new Integer(300);
```

Solution

Exercise

Identify the objects and classes

```
String s;
double delta = 1.89;
boolean equal = false;
```

```
Object object;
Integer count = new Integer(300);
```

Declaration

The process of associating an object variable name with an object type (class).

```
String s;
ArrayList list;
Scanner s;
StringTokenizer st;
```

Instantiation

The process of creating a new instance of a class.

This is done in order to use the services of a class.

The **new** keyword is used.

```
String s = new String();
ArrayList list = new ArrayList();
Scanner s = new Scanner();
StringTokenizer st = new StringTokenizer();
```

Initialisation

The process of assigning state (values) to an instance of a class.

```
String s = new String(“Clear Tape”);
ArrayList list = new ArrayList(10);
Scanner s = new Scanner(“data.txt”);
StringTokenizer st = new StringTokenizer(“;”);
```

Exercise

Distinguish between instantiation, declaration and initialisation

```
String s;
s = “Clear Tape”;
ArrayList stationery;
stationery = new ArrayList();
new File(“stocks.txt”);
String message = new String(“Out of Stock”);
```

Exercise

Solution

Distinguish between instantiation, declaration and initialisation

```
String s; // Declaration
s = “Clear Tape”; // Initialisation
ArrayList stationery; // Declaration
stationery = new ArrayList(); //Instantiation
new File(“stocks.txt”); //Instantiation, no Declaration!
String message = new String(“Out of Stock”); //all 3
```

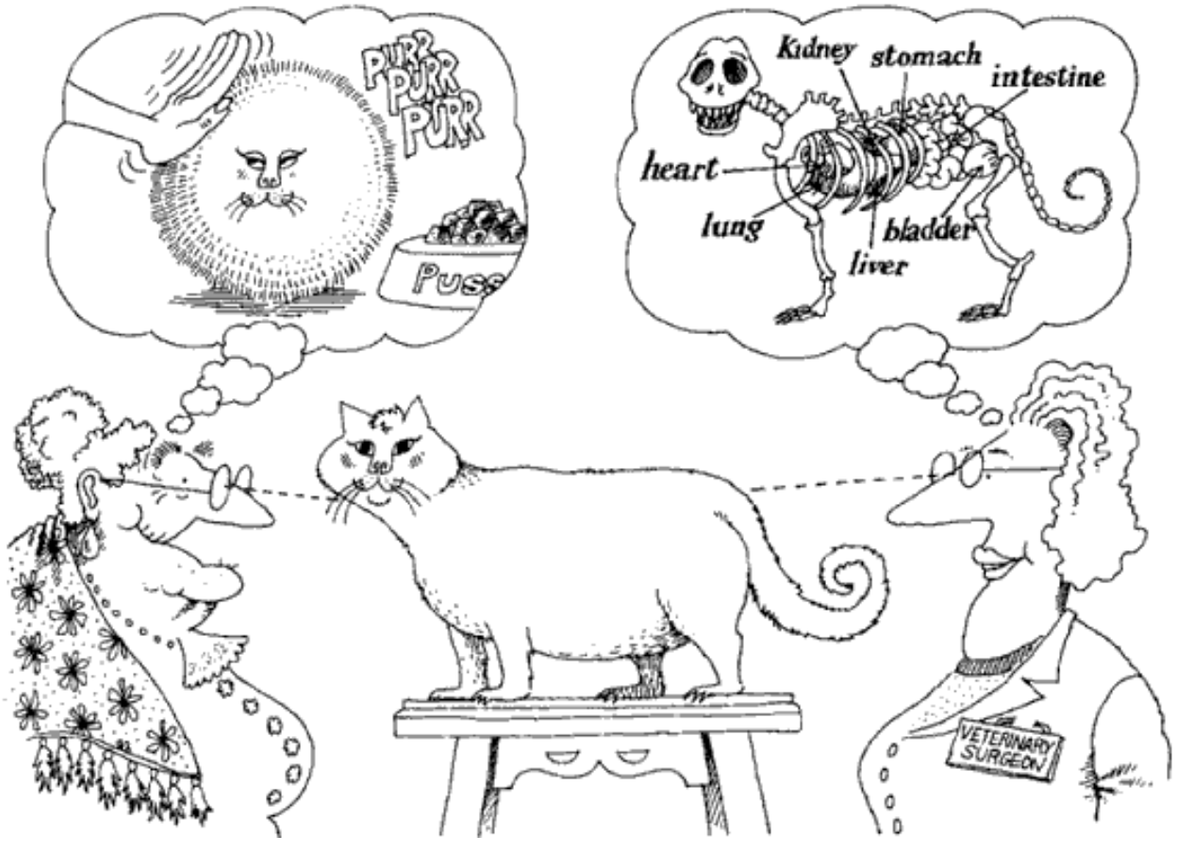
Abstraction

"Abstraction is one of the fundamental ways that we as humans cope with complexity." (Booch, 1994).

- Recognition of similarities
- Emphasis on significant details
- Independent of implementing mechanism

Abstraction

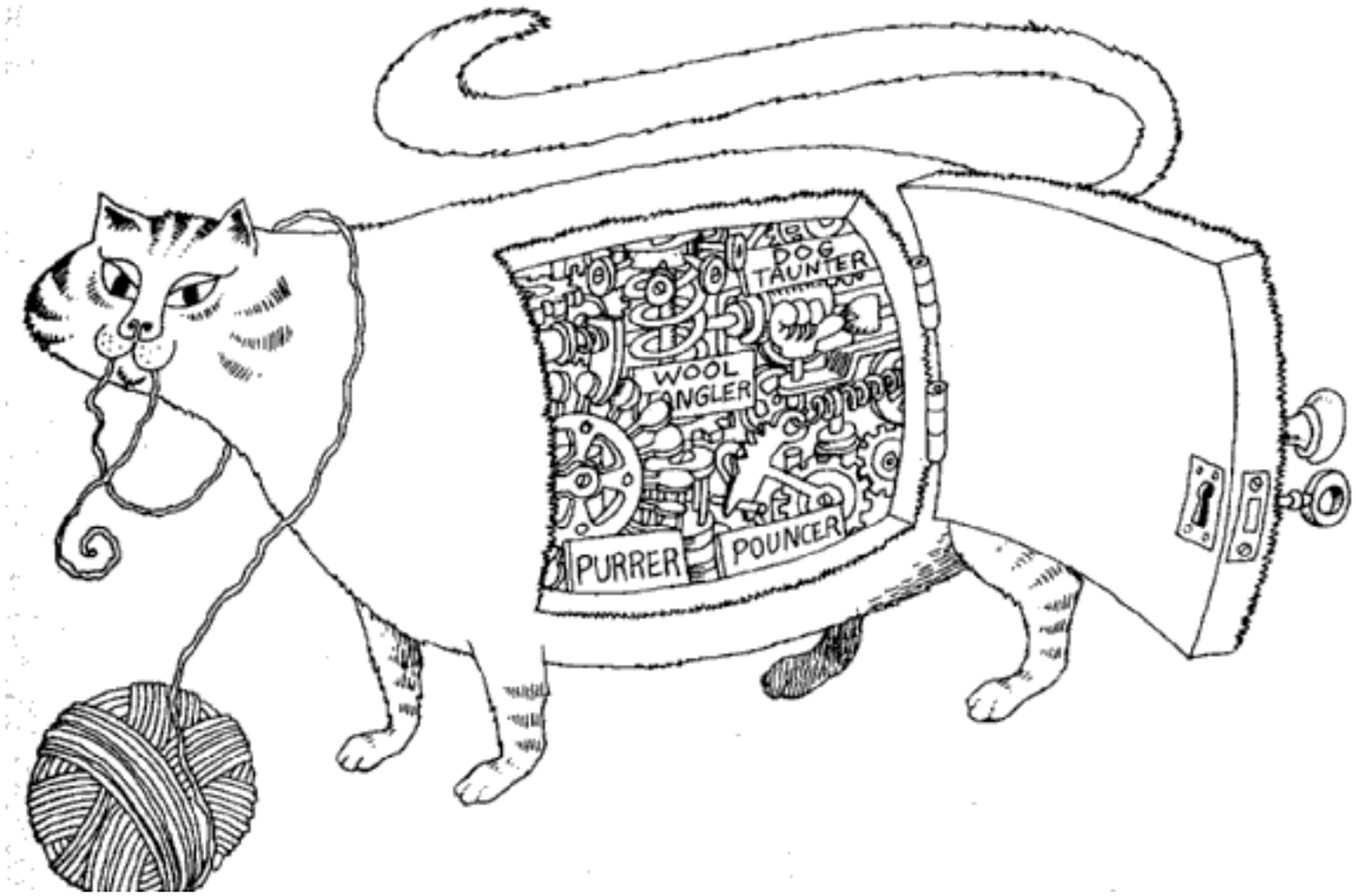
An abstraction denotes the **essential** characteristics of an object that **distinguish** it from all other kinds of objects and thus provide crisply defined **conceptual boundaries**, relative to the perspective of the viewer.



Observable behaviour

Encapsulation

Encapsulation focuses upon the implementation that gives rise to observable behaviours.



Hides the details (implementation) that gives rise to observable behaviour

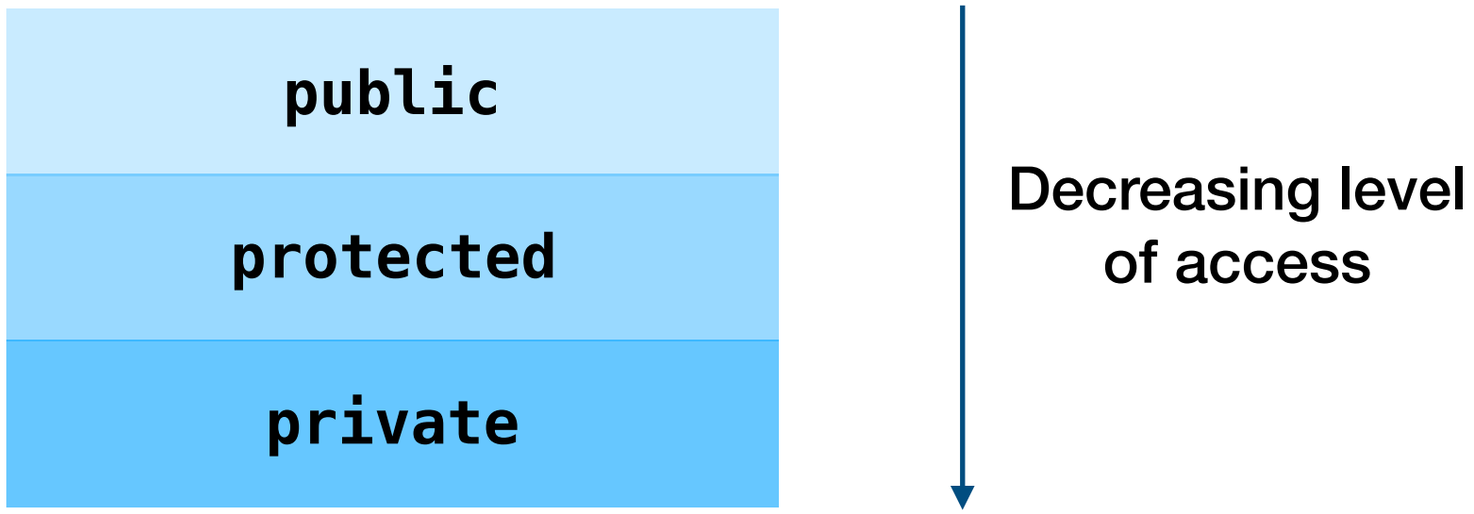
Information Hiding

Encapsulation is most often achieved through information hiding.

Information Hiding is the process of hiding all the secrets of an object that do not contribute to its essential characteristics:

- the structure of an object is hidden
- the implementation of its methods is hidden

Access Modifiers



- Public: A declaration that is accessible to all clients
- Protected: A declaration that is accessible only to the class itself, its subclasses, and its friends
- Private: A declaration that is accessible only to the class itself and its friends

Interface vs Implementation

The interface of a class provides its outside view and therefore emphasizes the abstraction while hiding its structure and the secrets of its behavior.

This interface primarily consists of the declarations of all the operations applicable to instances of this class, but it may also include the declaration of other classes, constants, variables, and exceptions as needed to complete the abstraction.

Interface vs Implementation

By contrast, the implementation of a class is its inside view, which encompasses the secrets of its behaviour.

The implementation of a class primarily consists of the implementation of all of the operations defined in the interface of the class.

Relationships

Classes, like objects, do not exist in isolation. Very often, an object-oriented program consists of a set of interacting objects whose classes are related in some way.

Relationships between classes are established to either:

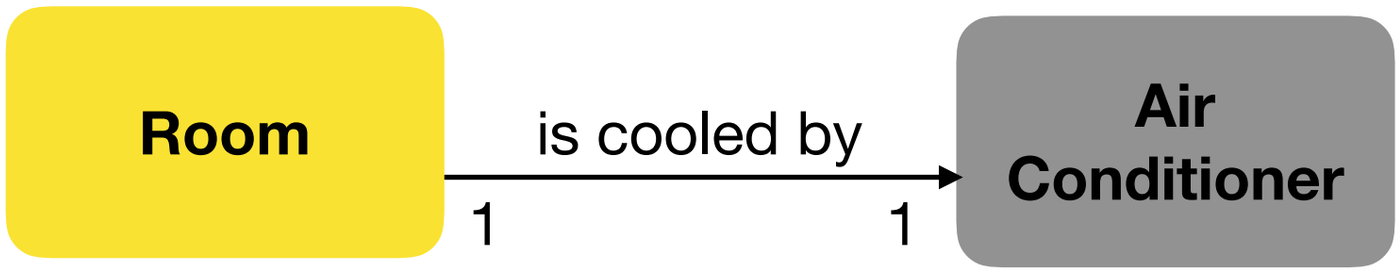
- Indicate some sort of sharing between the classes
- Indicate a semantic connection between the classes

Kinds of Relationships

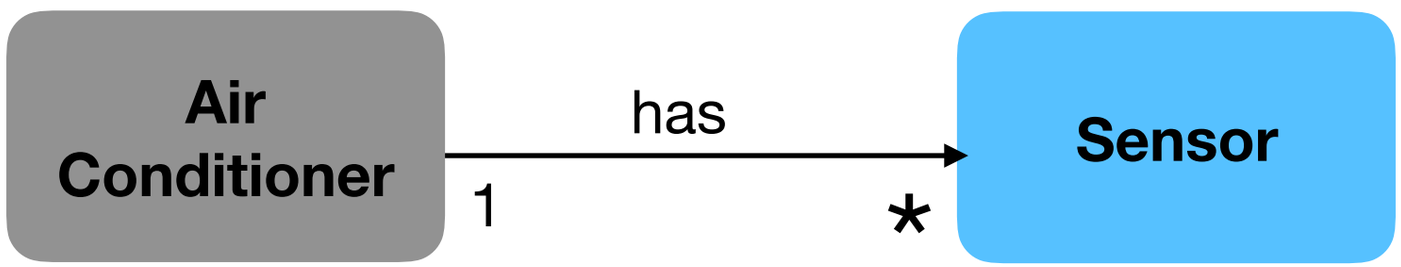
There are three basic kinds of relationships:

1. Association/Dependency (uses)
2. Generalisation/Specialisation (is-a)
3. Composition/Aggregation (part-of)

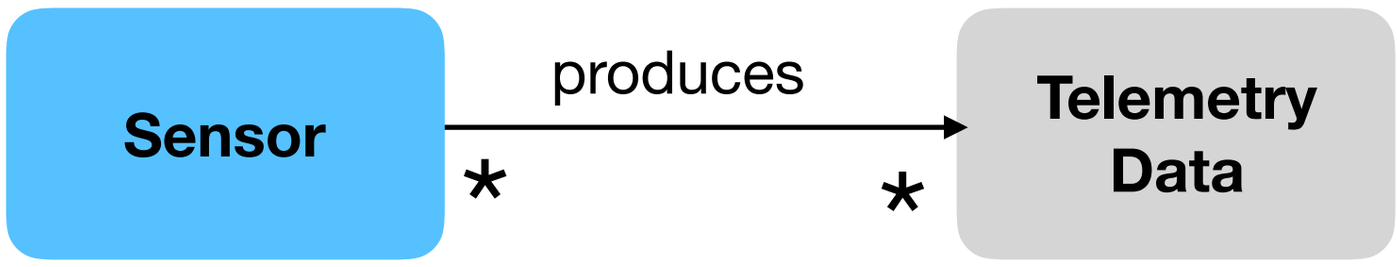
Example- Association



one-to-one association

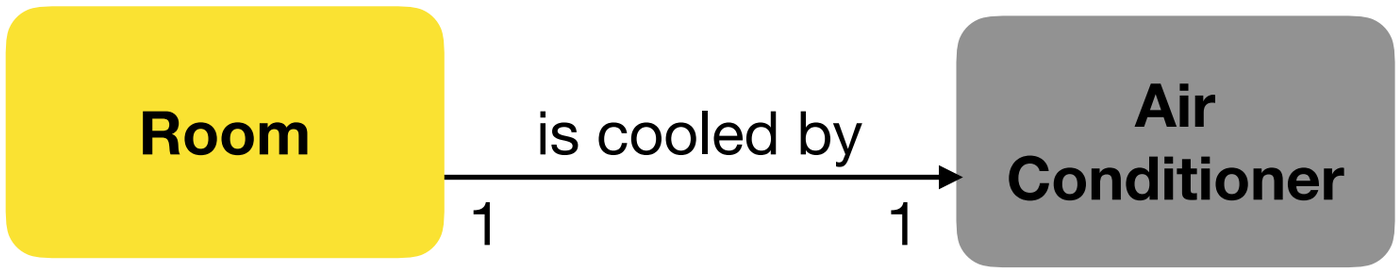


one-to-many association

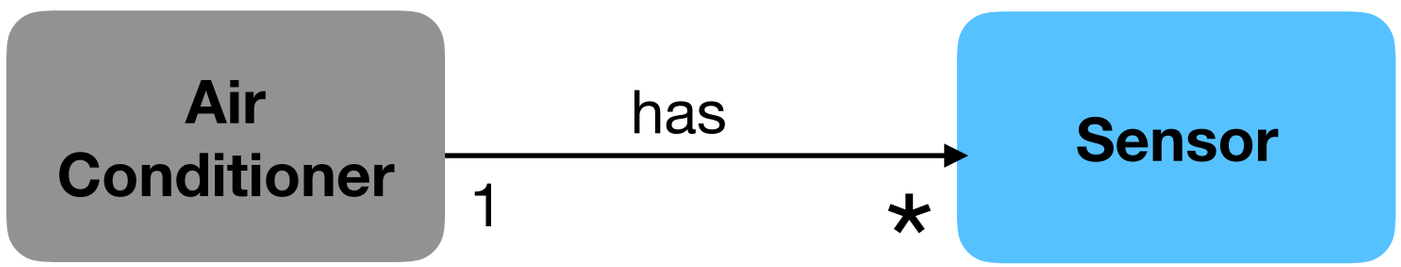


many-to-many association

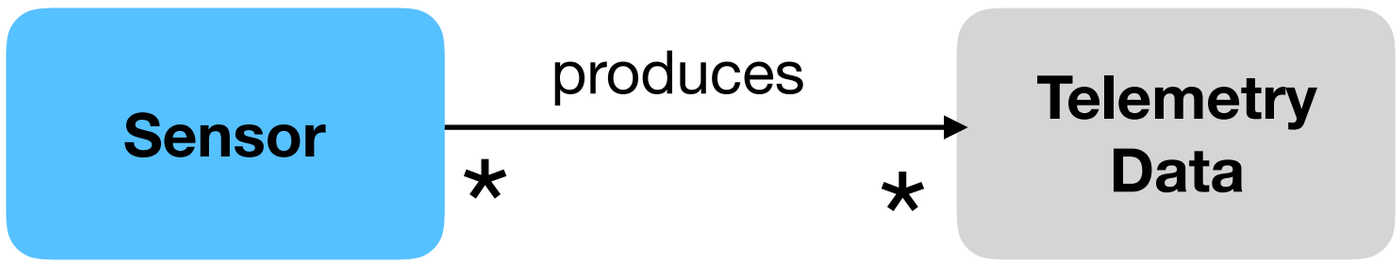
Example- Association



one-to-one association



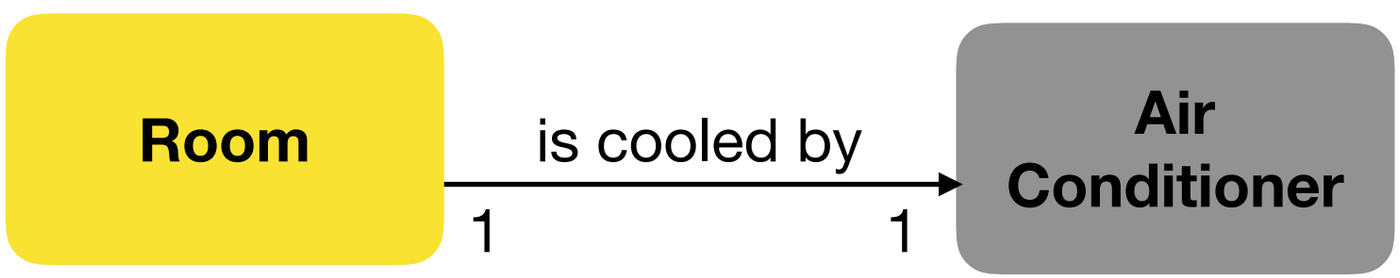
one-to-many association



many-to-many association

Example - Implementing Associations

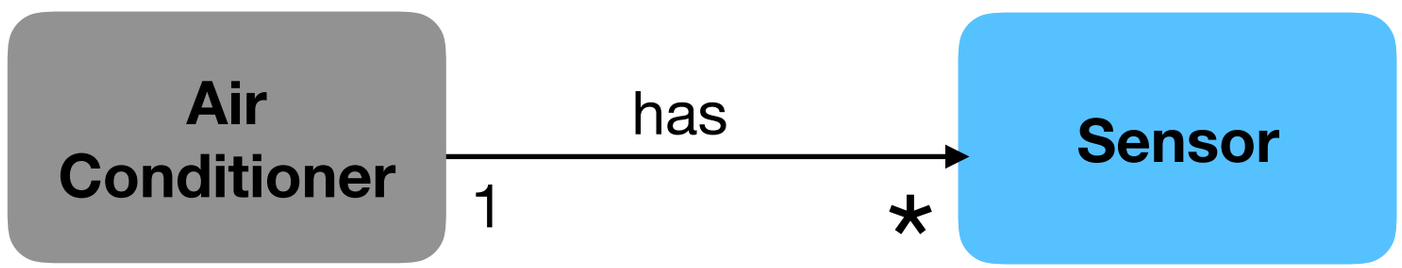
```
public class Room{
    private AirConditioner ac;
    public Room( ){
      ac = new AirConditioner( );
    }
}
```



one-to-one association

Example - Implementing Associations

```
public class AirConditioner{
    private Sensor[] sensors;
    public AirConditioner( ){
     sensors = new Sensor[3];
    }
}
```



one-to-many association

Exercise

Identify the objects and their associations in the following scenario.

A point-of-sale (POS) system is a computerised application used in a retail store to record sales and handle payments.

Solution

Exercise

Identify the objects first

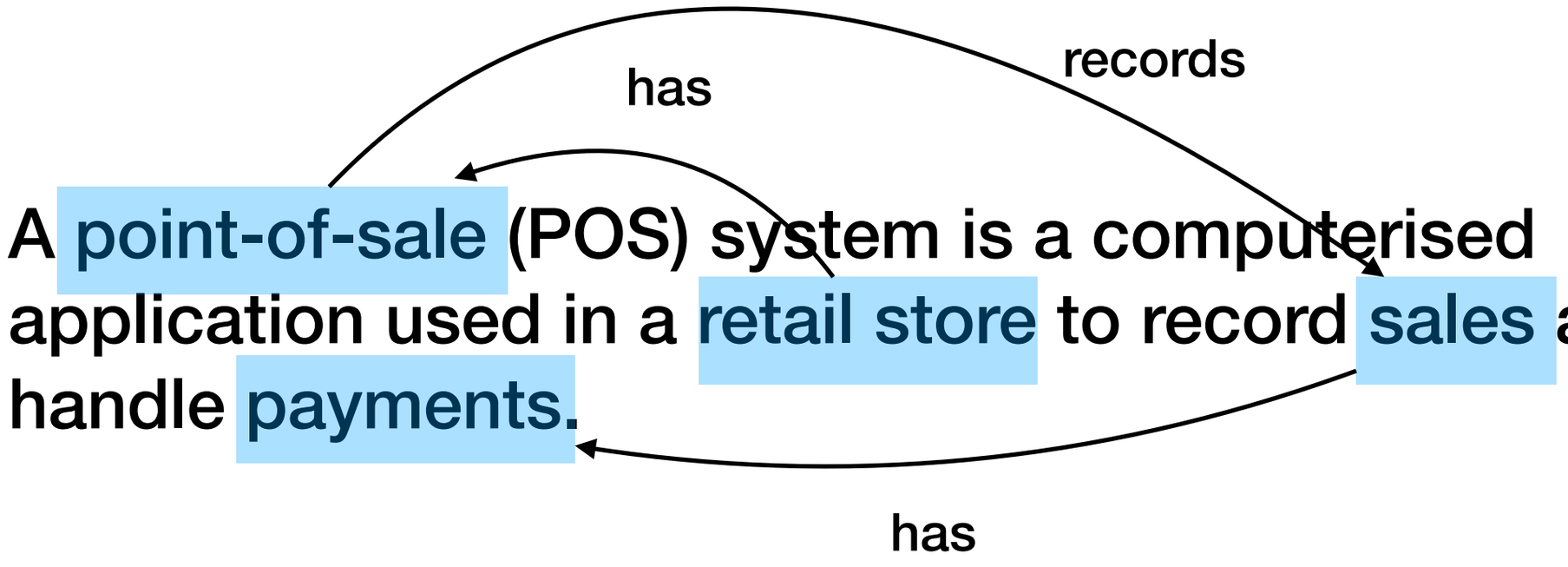
Objects

A point-of-sale (POS) system is a computerised application used in a retail store to record sales and handle payments.

Solution

Exercise

Next, identify the main associations between the objects.

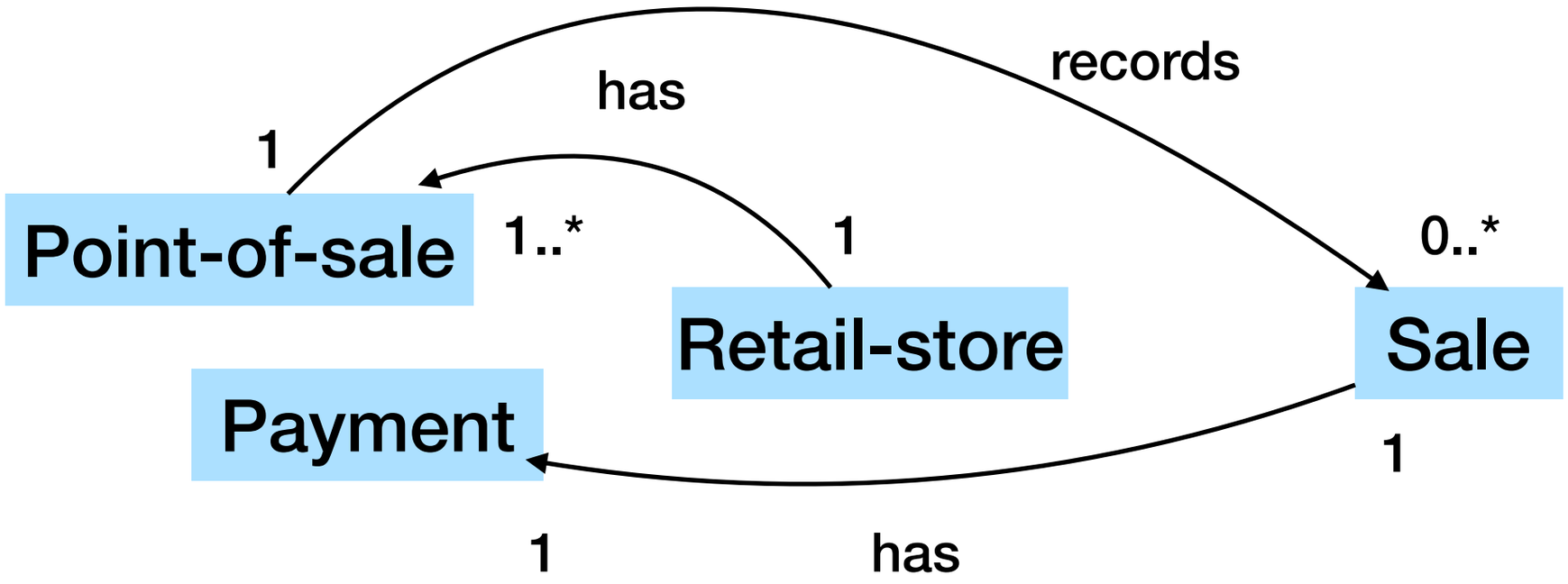


Associations

Solution

Exercise

Clean up and assign cardinalities.



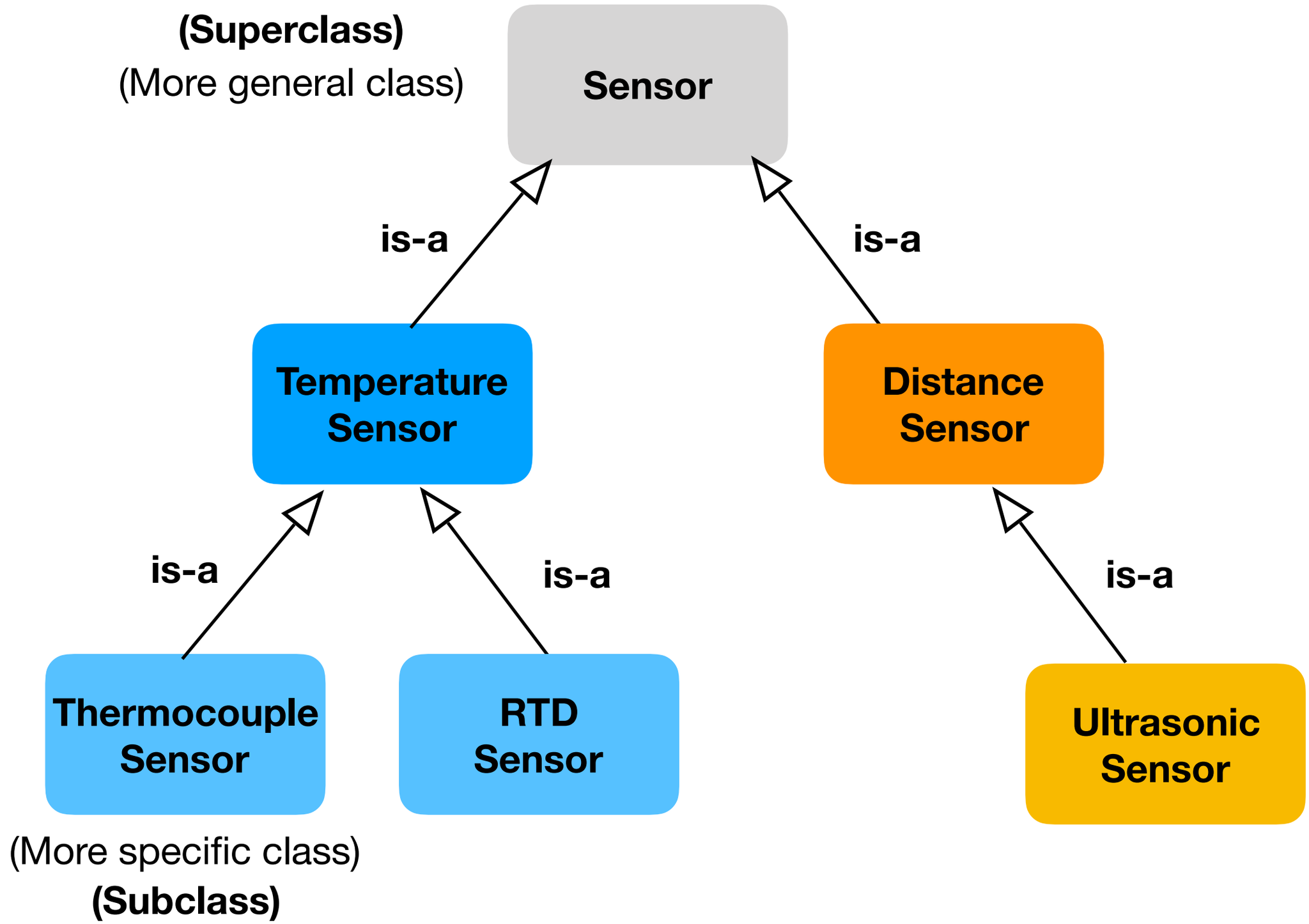
Cardinalities

Inheritance

Inheritance is a relationship among classes wherein one class shares the **structure** and/or **behaviour** defined in one (single inheritance) or more (multiple inheritance) other classes (Booch 1994).

A class from which another class inherits its structure and/or behaviour is called the **superclass**. A class that inherits from one or more classes is called a **subclass**

Example - Inheritance

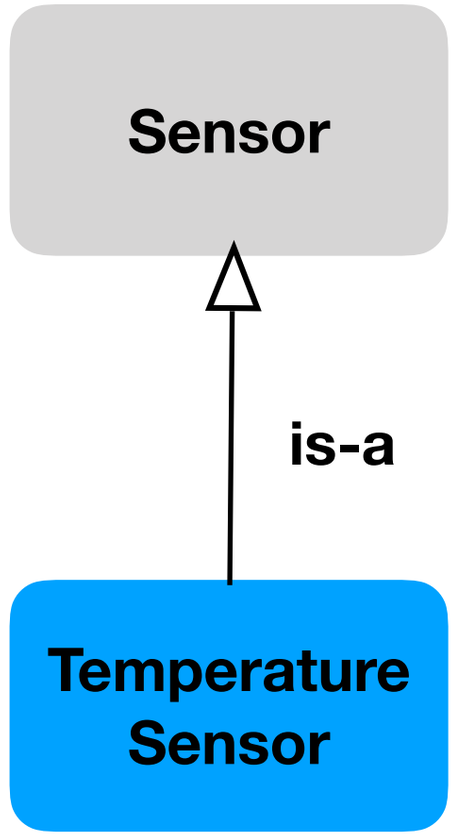


Example Implementing Inheritance

```
public class Sensor{
   private String unit;
   private double maxRangeValue;
   private double minRangeValue;
   private int responsivenessLapse;

   public Sensor(){
   }
   //accessors & mutators
}
```

```
public class TemperatureSensor extends Sensor{
}
```



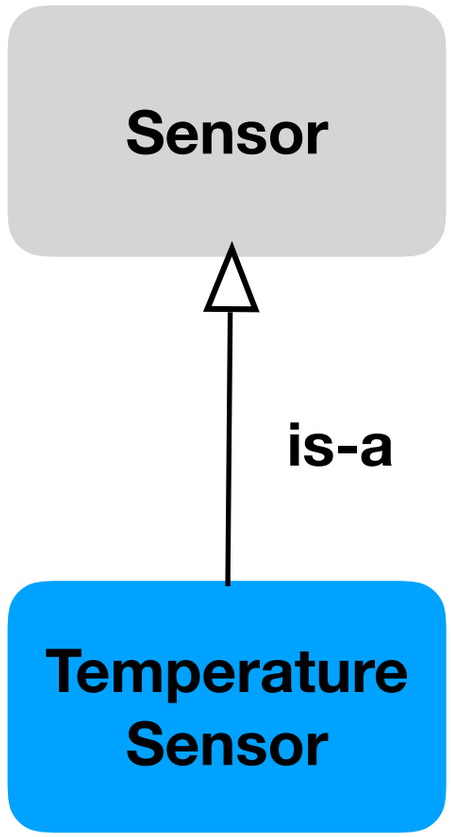

```
TemperatureSensor t = new TemperatureSensor();
t.setUnit();
t.getMinRangeValue();
t.getUnit();
t.unit = “Celsius”;
```

Client class

Example Implementing Inheritance

```
public class Sensor{
   protected String unit;
   protected double maxRangeValue;
   protected double minRangeValue;
   protected int responsivenessLapse;

   public Sensor(){
   }
   //accessors & mutators
}
```




```
public class TemperatureSensor extends Sensor{
}
```

```
TemperatureSensor t = new TemperatureSensor();
t.setUnit();
t.getMinRangeValue();
t.getUnit();
t.unit = “Celsius”;
```

Client class

Exercise

Identify the objects and their relationships in the following scenario.

Payments can be either cash or card payments. Card payments can be either debit card or credit card. Credit card payments are subject to a 4% fee.

Solution

Exercise

Identify the objects

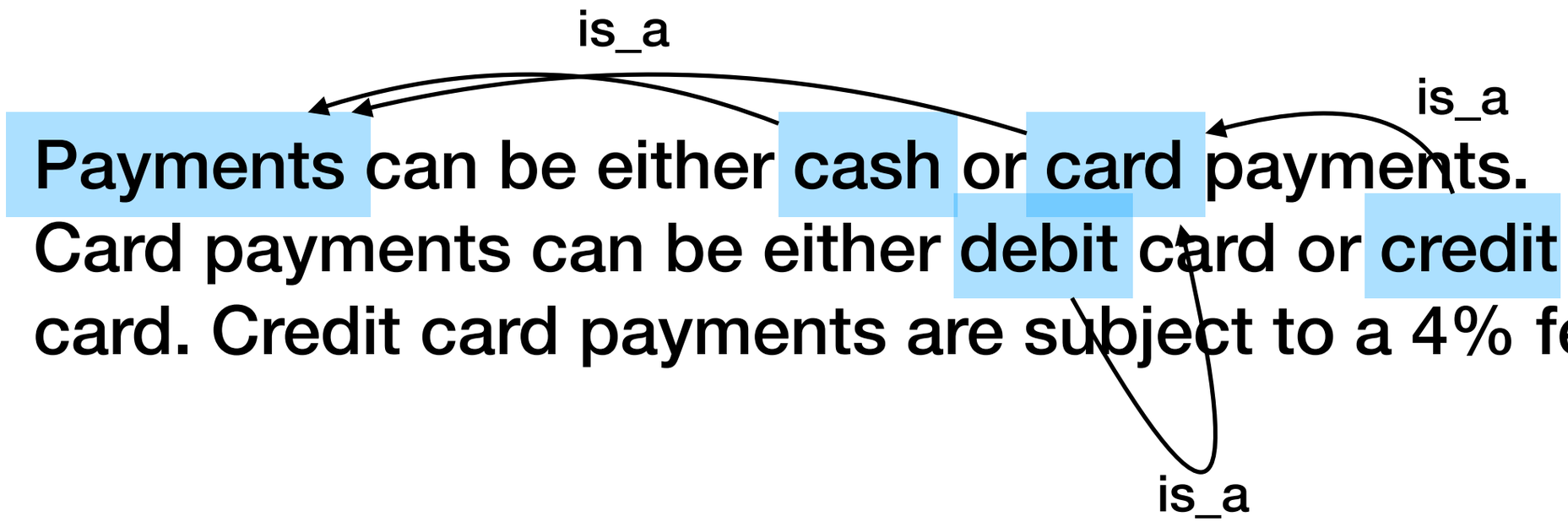
Payments can be either cash or card payments. Card payments can be either debit card or credit card. Credit card payments are subject to a 4% fee.

Objects

Exercise

Solution

Identify relationships

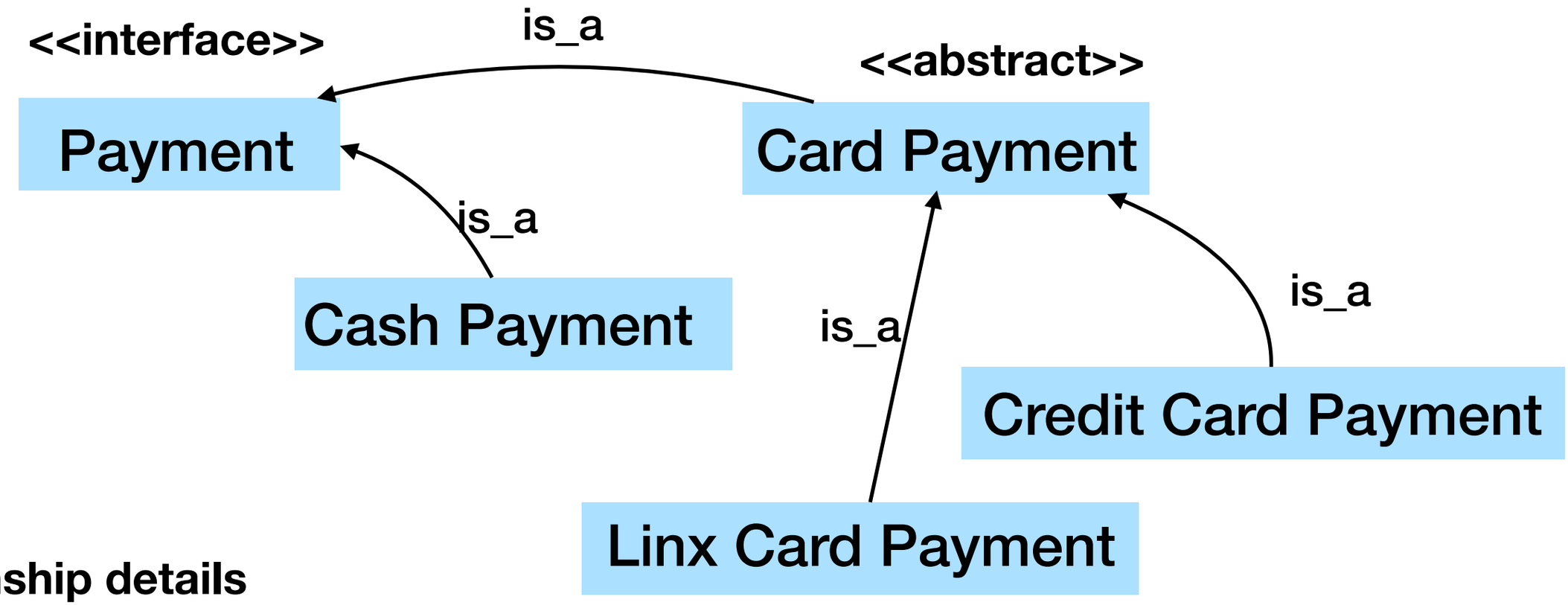


Relationships

Exercise

Solution

Fine tune



Relationship details

Polymorphic Objects

When the static type and the dynamic type of an object are different, that object is said to be **polymorphic.**

A polymorphic object exhibits different behaviour based on the differences between its static type and dynamic type.

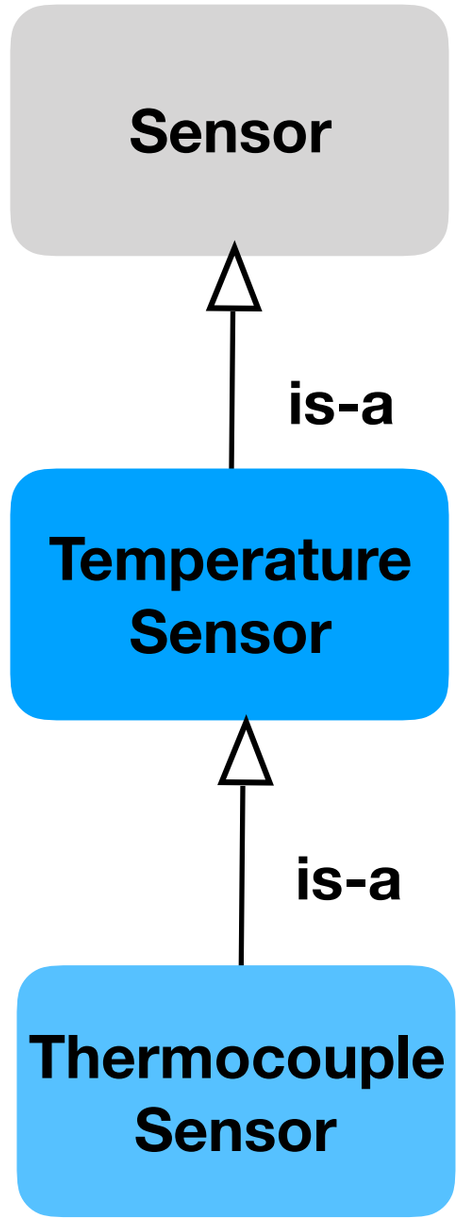
Example - Polymorphic Objects

```
Sensor ts = new ThermocoupleSensor();
//ST: Sensor,   DT: ThermocoupleSensor

TemperatureSensor ts = new TemperatureSensor();
//ST: TemperatureSensor,   DT: TemperatureSensor

Object o = new ThermocoupleSensor();
//ST: Object,   DT: ThermocoupleSensor
```

(Only 2 objects are Polymorphic)



Abbreviations:
ST - Static type; DT - Dynamic type

References

- Booch, Grady. (1988) OBJECT-ORIENTED ANALYSIS AND DESIGN
- Mohan, Permanand (2013) FUNDAMENTALS OF OBJECT-ORIENTED PROGRAMMING IN JAVA