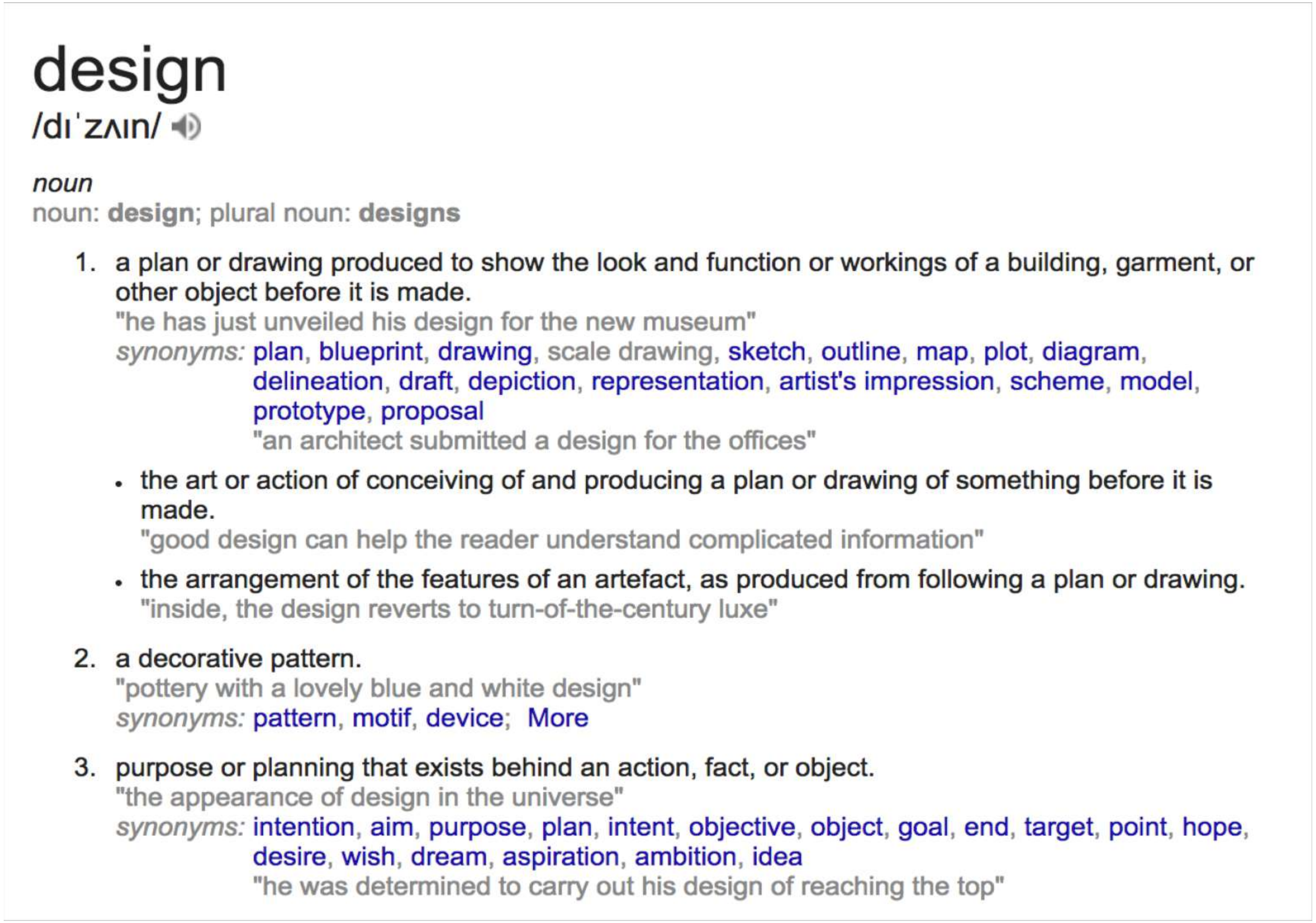
Object Oriented Design

Design before you build.



design

/dɪˈzʌɪn/

noun

noun: **design**; plural noun: **designs**

1. a plan or drawing produced to show the look and function or workings of a building, garment, or other object before it is made.
   "he has just unveiled his design for the new museum"
   synonyms: plan, blueprint, drawing, scale drawing, sketch, outline, map, plot, diagram, delineation, draft, depiction, representation, artist's impression, scheme, model, prototype, proposal
   "an architect submitted a design for the offices"
   - the art or action of conceiving of and producing a plan or drawing of something before it is made.
     "good design can help the reader understand complicated information"
   - the arrangement of the features of an artefact, as produced from following a plan or drawing.
     "inside, the design reverts to turn-of-the-century luxe"
2. a decorative pattern.
   "pottery with a lovely blue and white design"
   synonyms: pattern, motif, device; More
3. purpose or planning that exists behind an action, fact, or object.
   "the appearance of design in the universe"
   synonyms: intention, aim, purpose, plan, intent, objective, object, goal, end, target, point, hope, desire, wish, dream, aspiration, ambition, idea
   "he was determined to carry out his design of reaching the top"

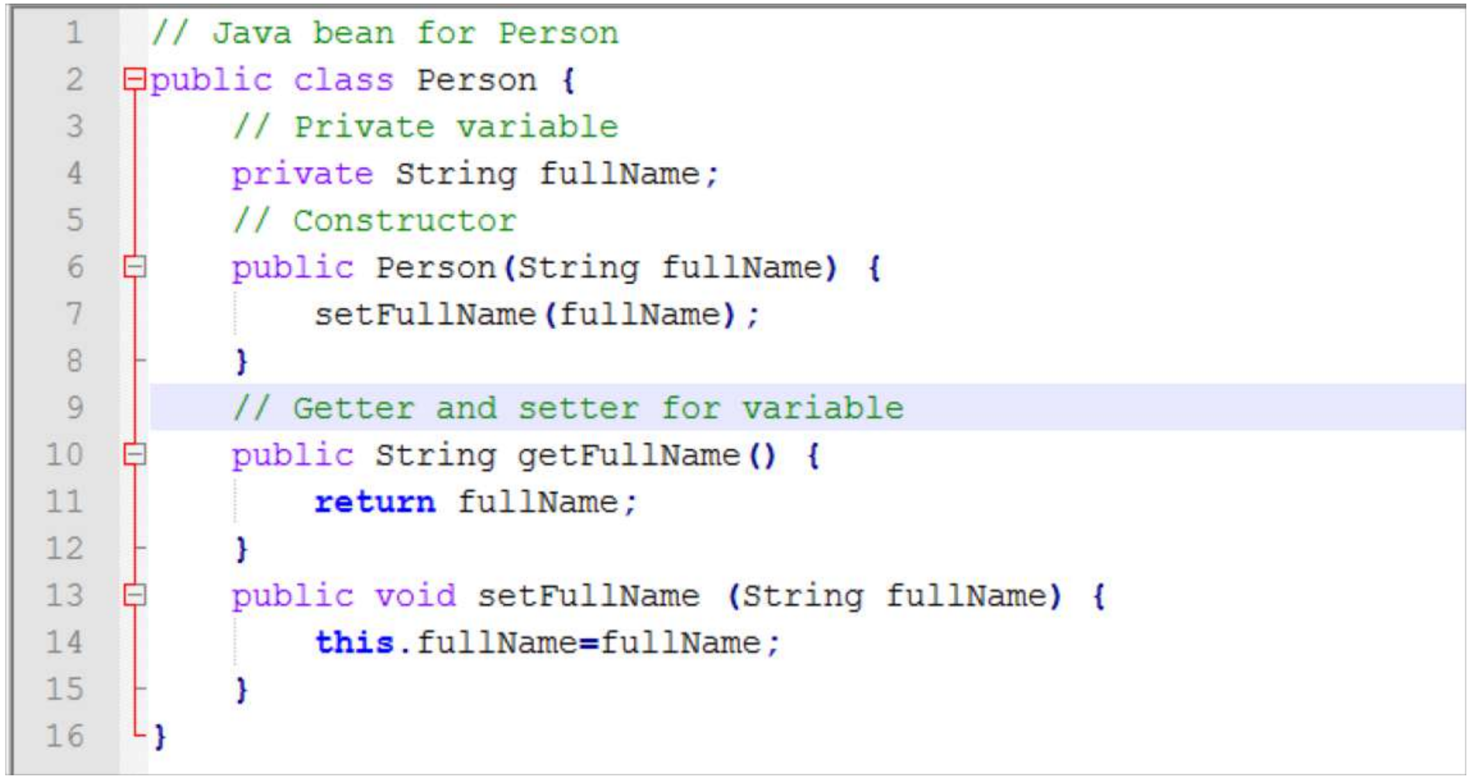
Concepts

- Object/Class
  - Class provide an abstraction for a certain thing.
  - Object is an instance of a class and is a tangible thing
- Encapsulation :- hide implementation and internal state of an object
- Inheritance: Enforces parent to child relationship and child acquired properties of parent
- Polymorphism & Overloading
- Binding

Class & Object

- Class
  - Defines properties and methods
  - Provides abstract notion for the developer to think in
  - Stateless
- Object
  - Instance of class
  - Tangible and has state
  - Has a unique identity

Class Showing Encapsulation



```java
// Java bean for Person
public class Person {
    // Private variable
    private String fullName;
    // Constructor
    public Person(String fullName) {
        setFullName(fullName);
    }
    // Getter and setter for variable
    public String getFullName() {
        return fullName;
    }
    public void setFullName (String fullName) {
        this.fullName=fullName;
    }
}
```

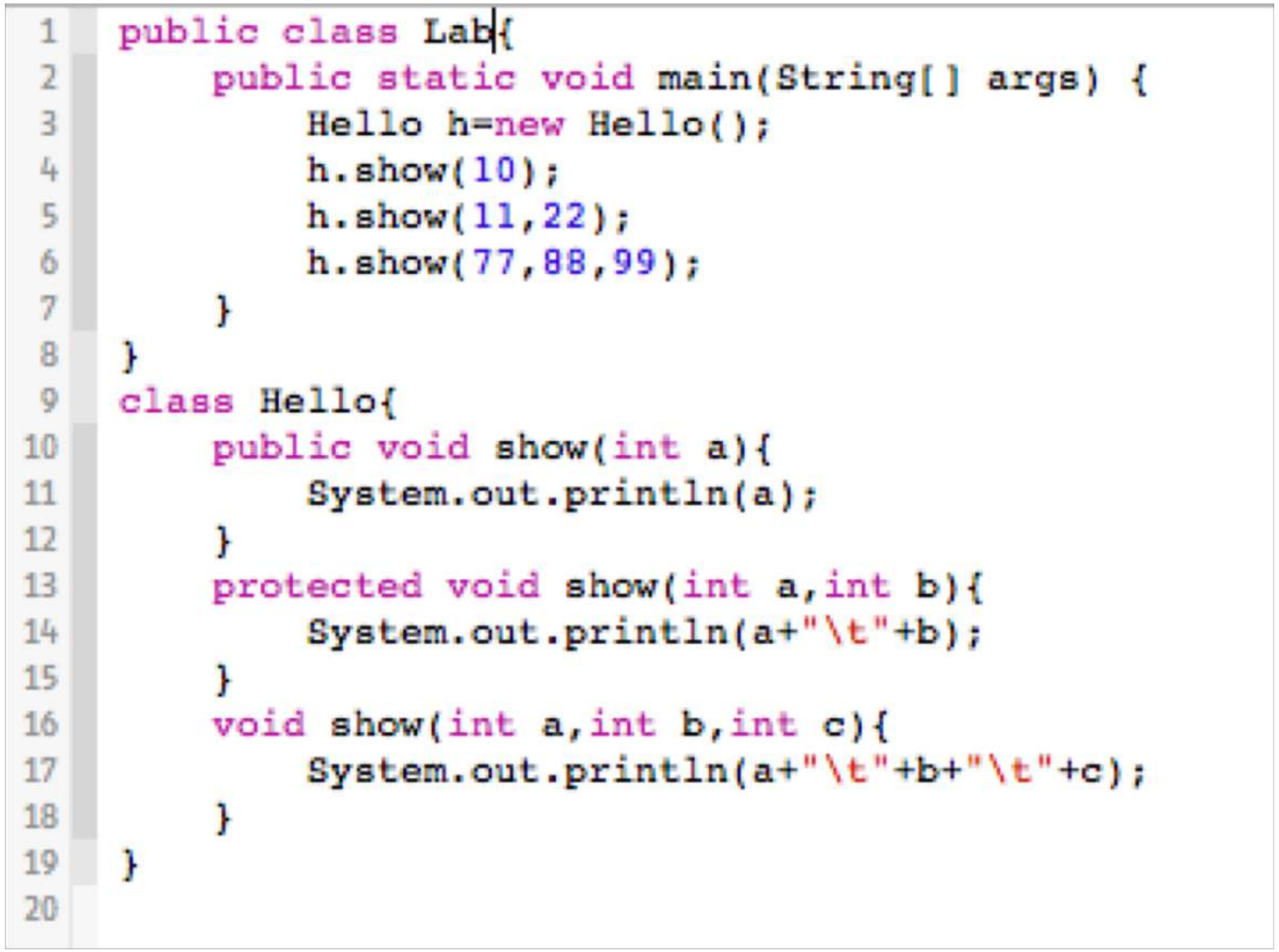
Polymorphism and Overloading

- Polymorphism:- Means to have many forms
- This is achieved by allowing multiple behaviors from the same function.
- This can be achieved by changing the number of parameters in function overloading or changing the behavior in child classes overriding

Binding

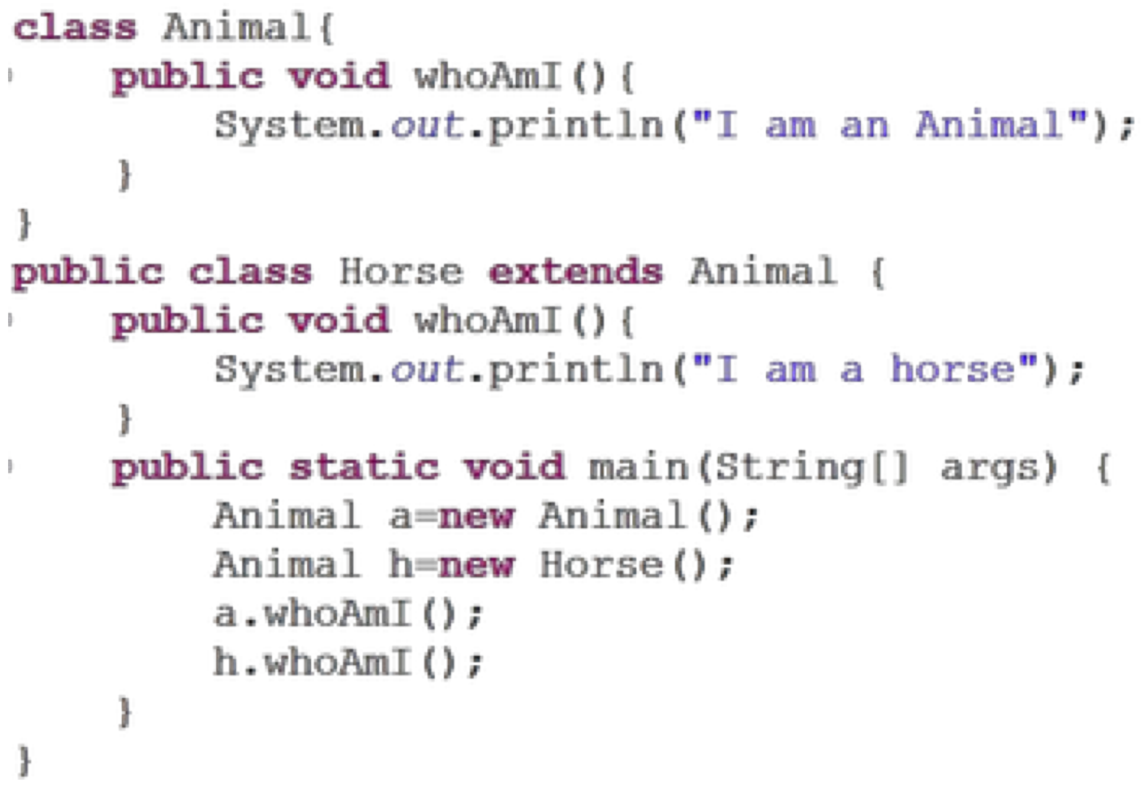
- Refers to linking procedure call to the object
- Static binding: Known during compile time: Method overloading is a a good example
- Late binding: Type is known only during runtime. Method overriding is an example

Overloading



```
public class Lab{
	public static void main(String[] args) {
		Hello h=new Hello();
		h.show(10);
		h.show(11,22);
		h.show(77,88,99);
	}
}
class Hello{
	public void show(int a){
		System.out.println(a);
	}
	protected void show(int a,int b){
		System.out.println(a+"\t"+b);
	}
	void show(int a,int b,int c){
		System.out.println(a+"\t"+b+"\t"+c);
	}
}
```

Overriding



```
class Animal{
    public void whoAmI(){
        System.out.println("I am an Animal");
    }
}
public class Horse extends Animal {
    public void whoAmI(){
        System.out.println("I am a horse");
    }
    public static void main(String[] args) {
        Animal a=new Animal();
        Animal h=new Horse();
        a.whoAmI();
        h.whoAmI();
    }
}
```

Why OOP

1

Reusability: Code can be reused in multiple applications

2

Privacy: prevents data loss in large programs {Power of encapsulation and data hiding}

3

Documentation: Helps in maintainability of the code

Next session

- What we will cover.
  - Terminologies
  - Notations
  - Introductions to design patterns

Terminologies in design

1

Conceptual model: captures the concept of the problem domain

- Independent of implementation details

2

Use case: Conveys how the system should interact with users to achieve a goal

3

Sequence diagrams: Shows sequences for a certain use case

UML

1

General purpose modeling language intended to provide a standard way to visualize the system design

2

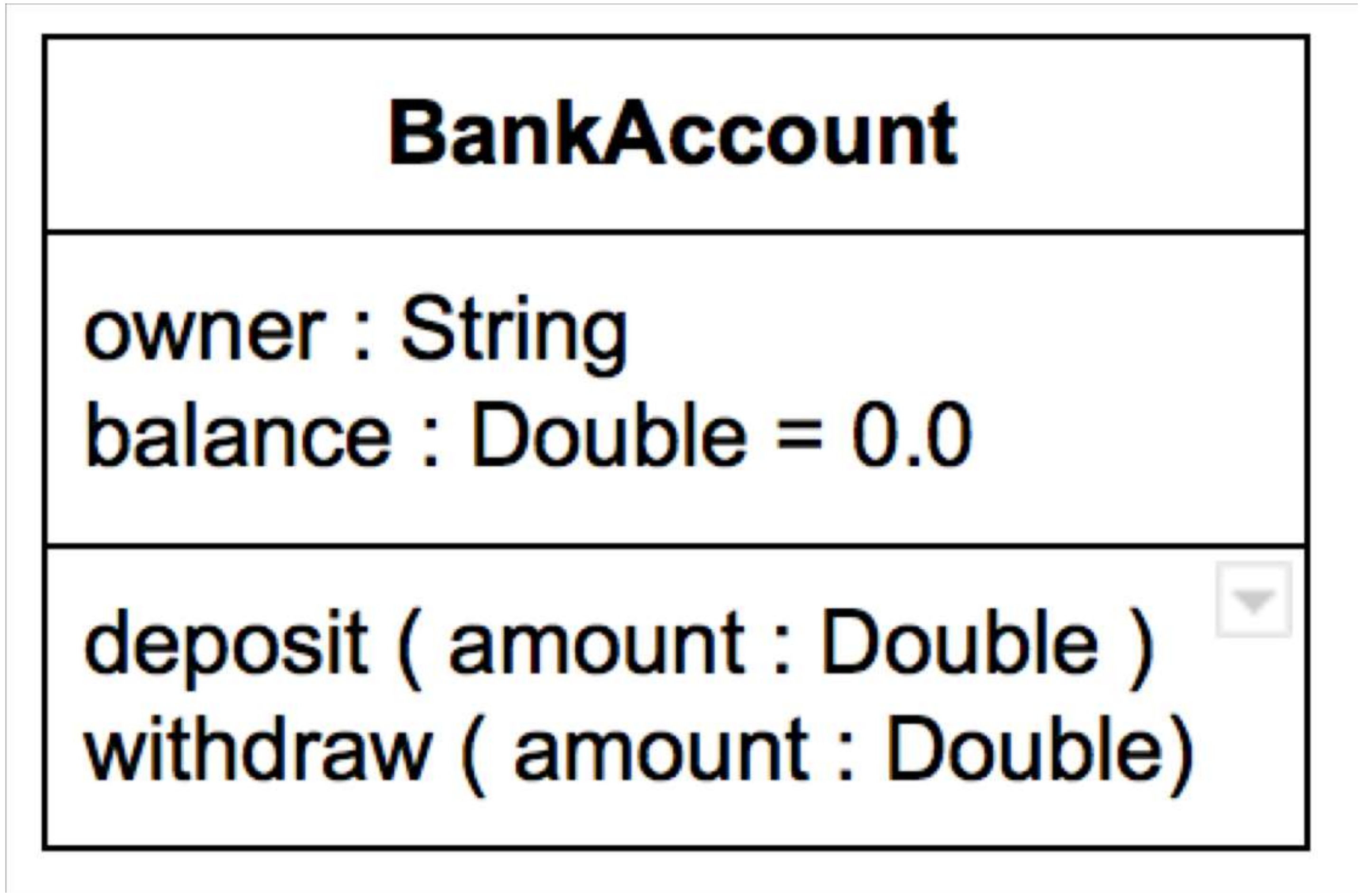
Can be used to show system design and behavior without code.

3

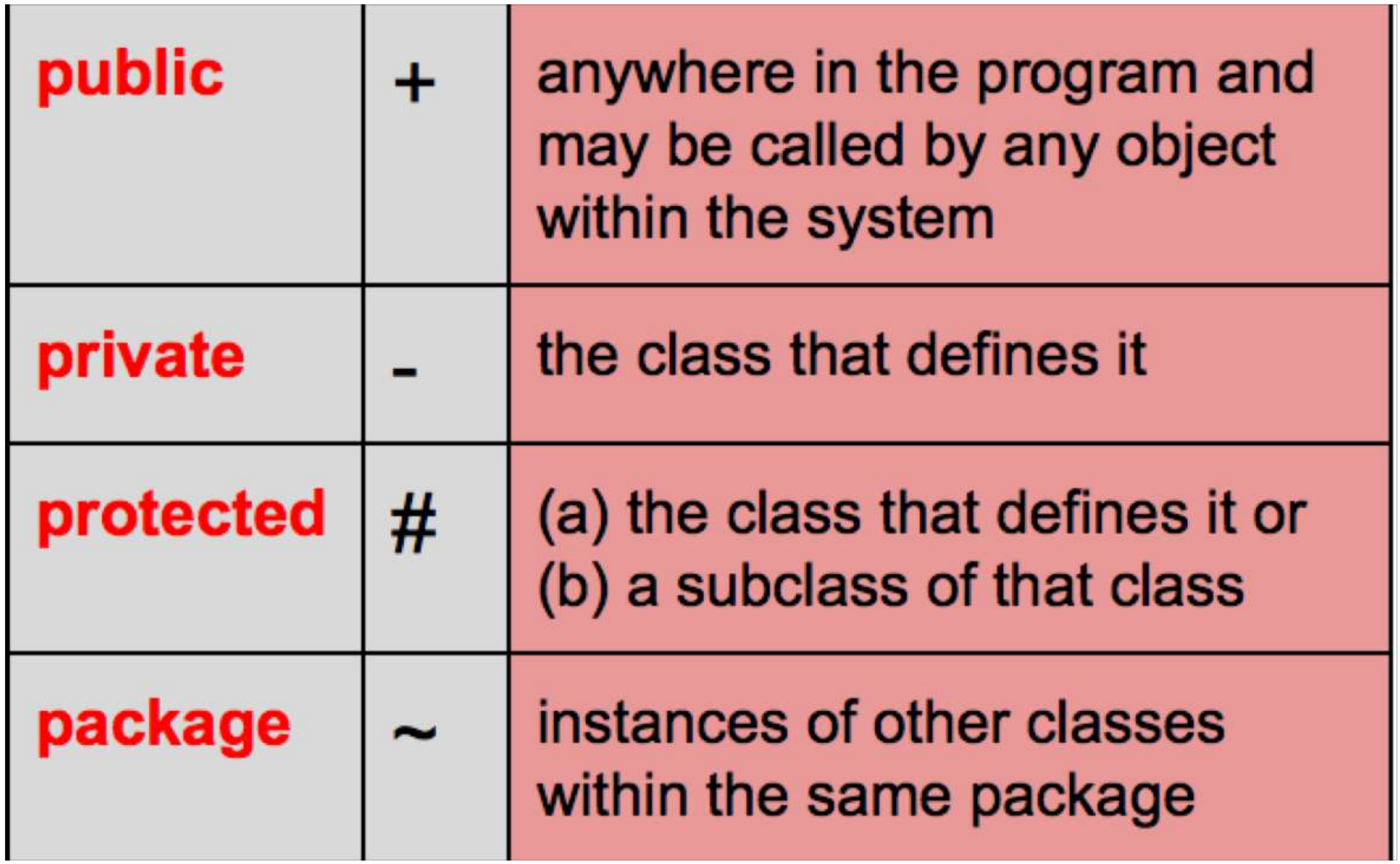
Types:

- Structure diagrams
- Behavior diagrams

Structure diagrams

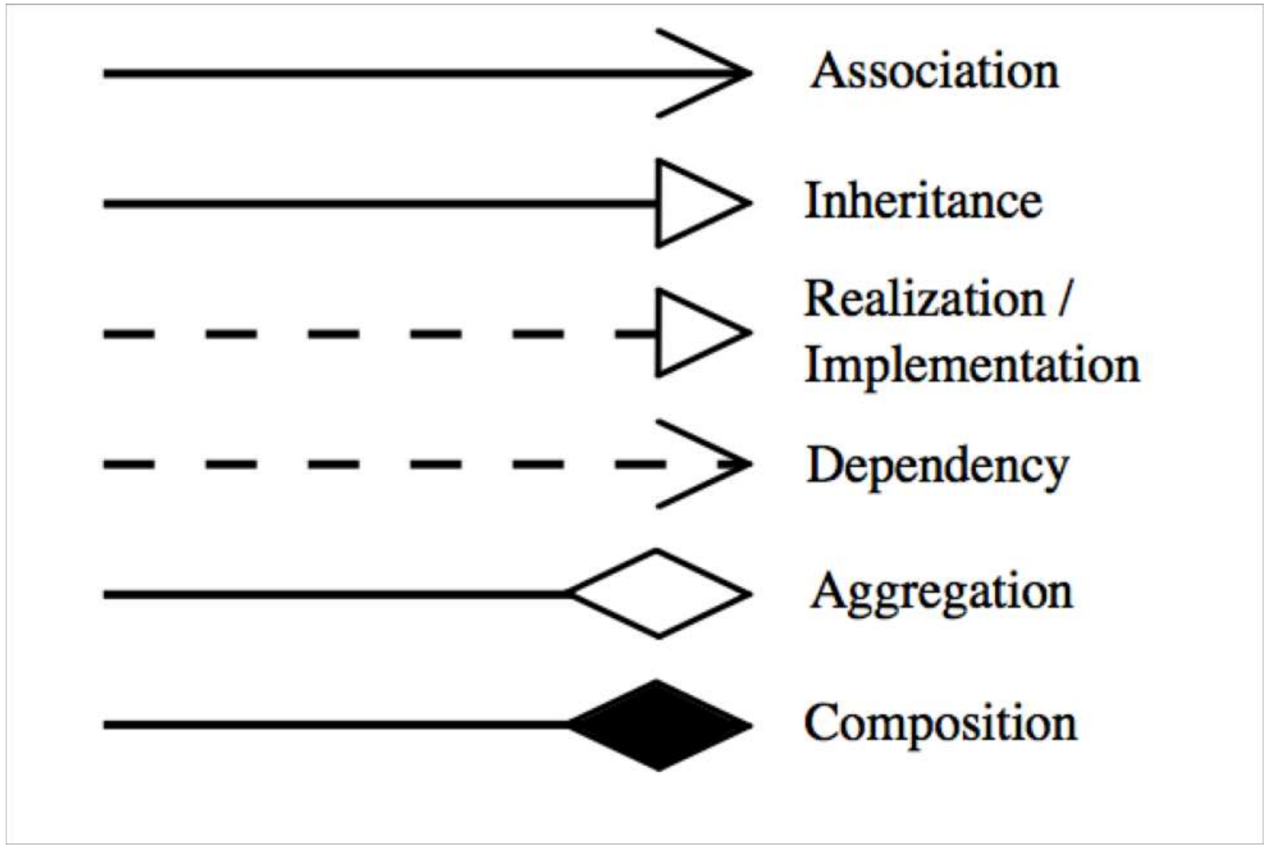


Visibility

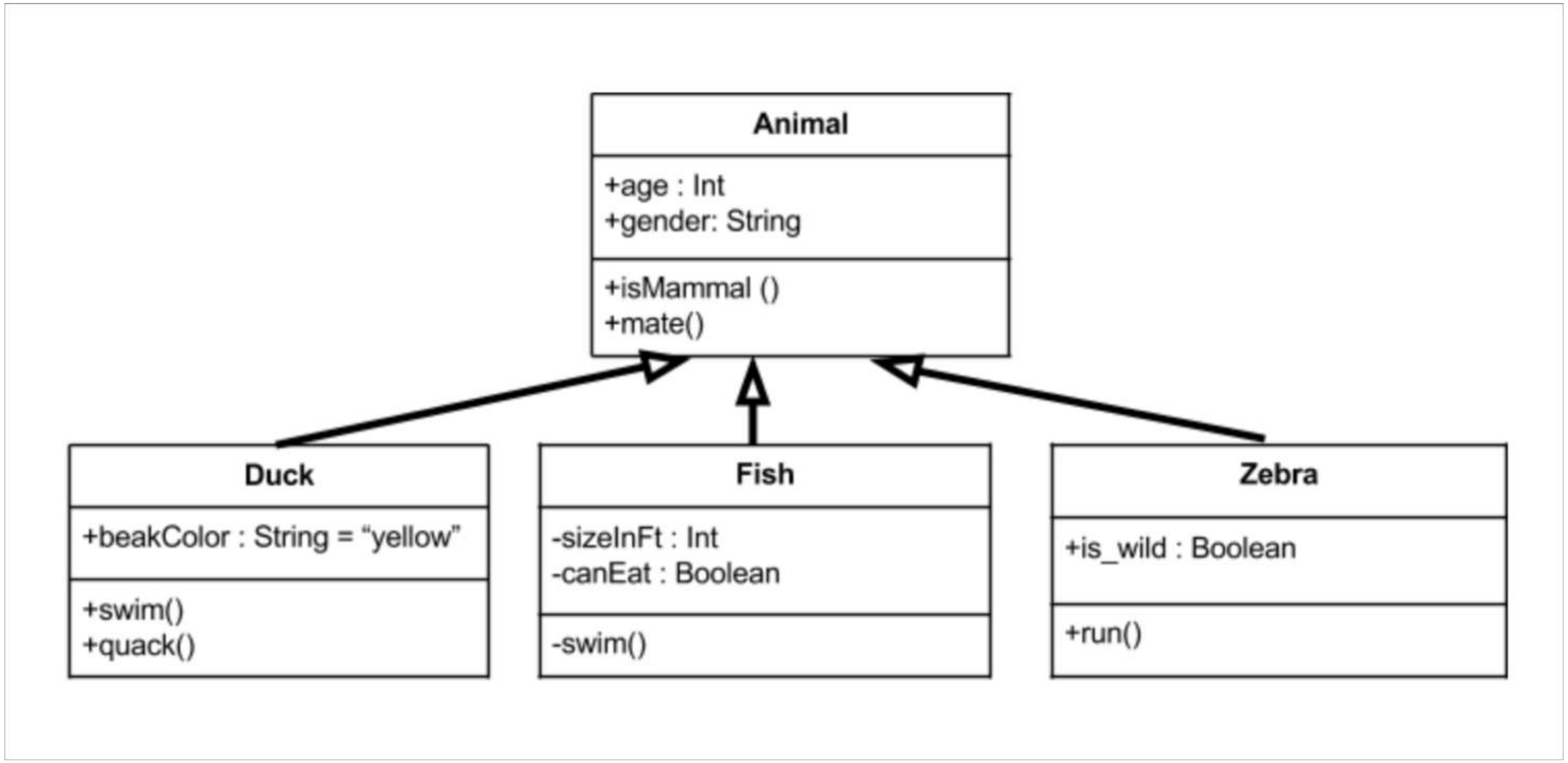


public	+	anywhere in the program and may be called by any object within the system
private	-	the class that defines it
protected	#	(a) the class that defines it or (b) a subclass of that class
package	~	instances of other classes within the same package

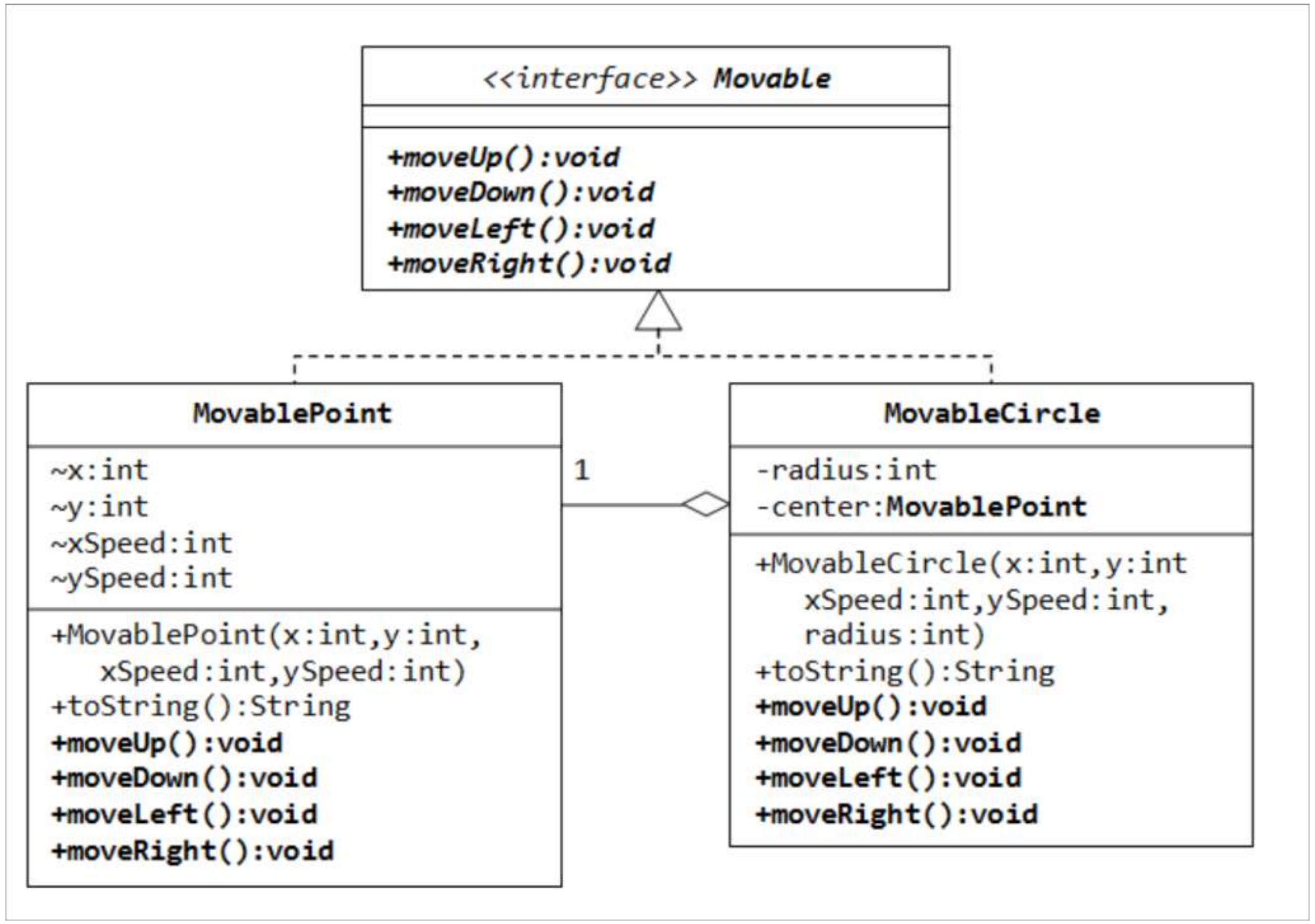
Relationships



Inheritance



Realization/implementation



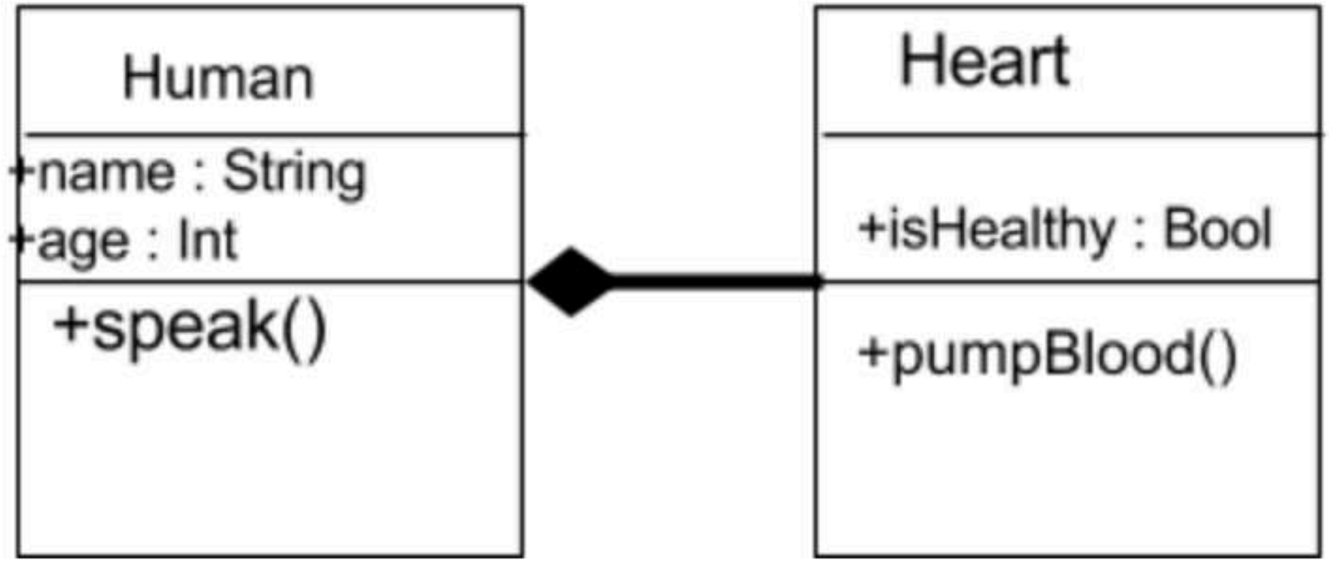
Aggregations

- Also called has a or is part of relationship
- Unidirectional
- E.g Wallet and money

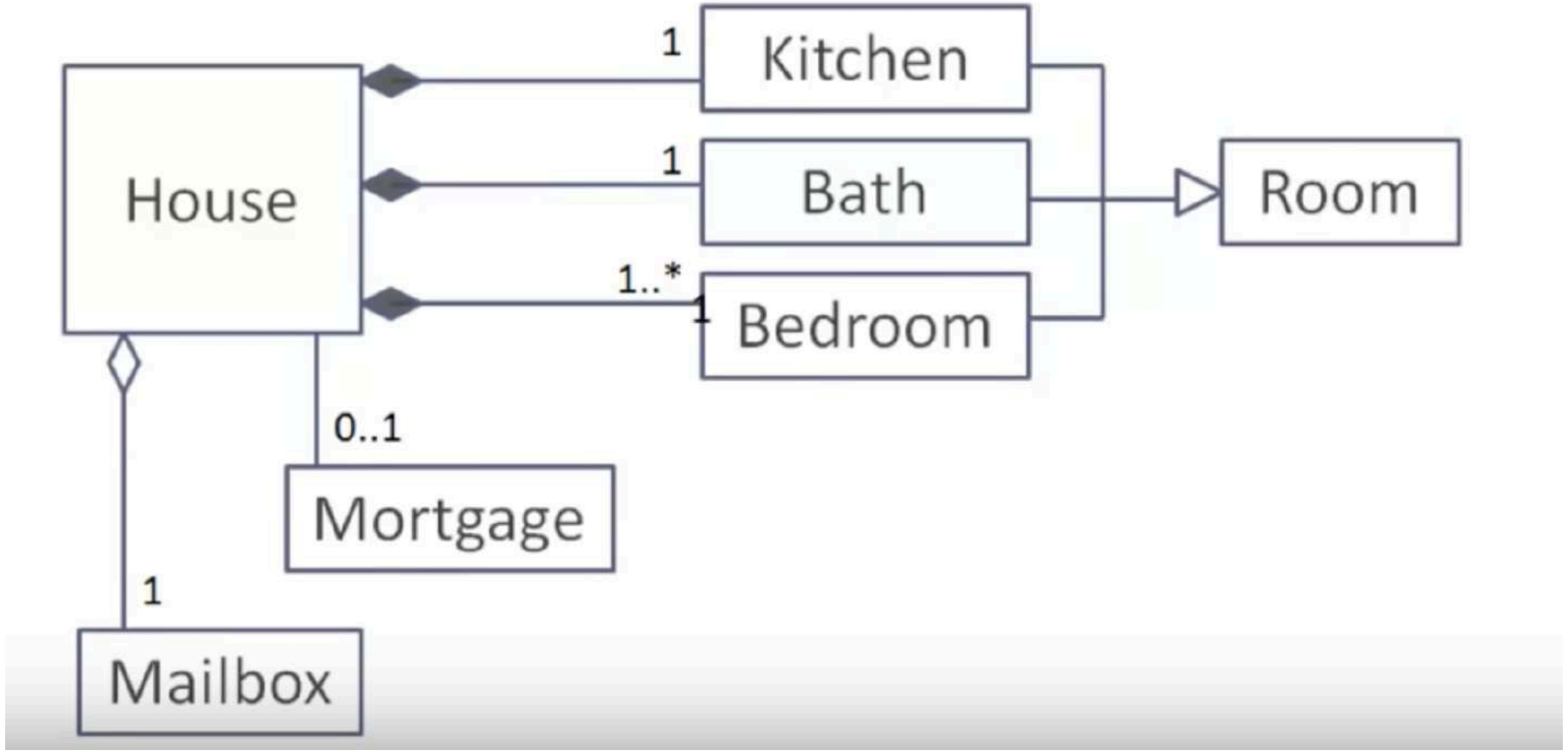
Compositions

- Shows more dependency between the entities
- Restricted form of aggregation as both classes depend on each other
- They share their lifecycle
    - E.g Human & Heart

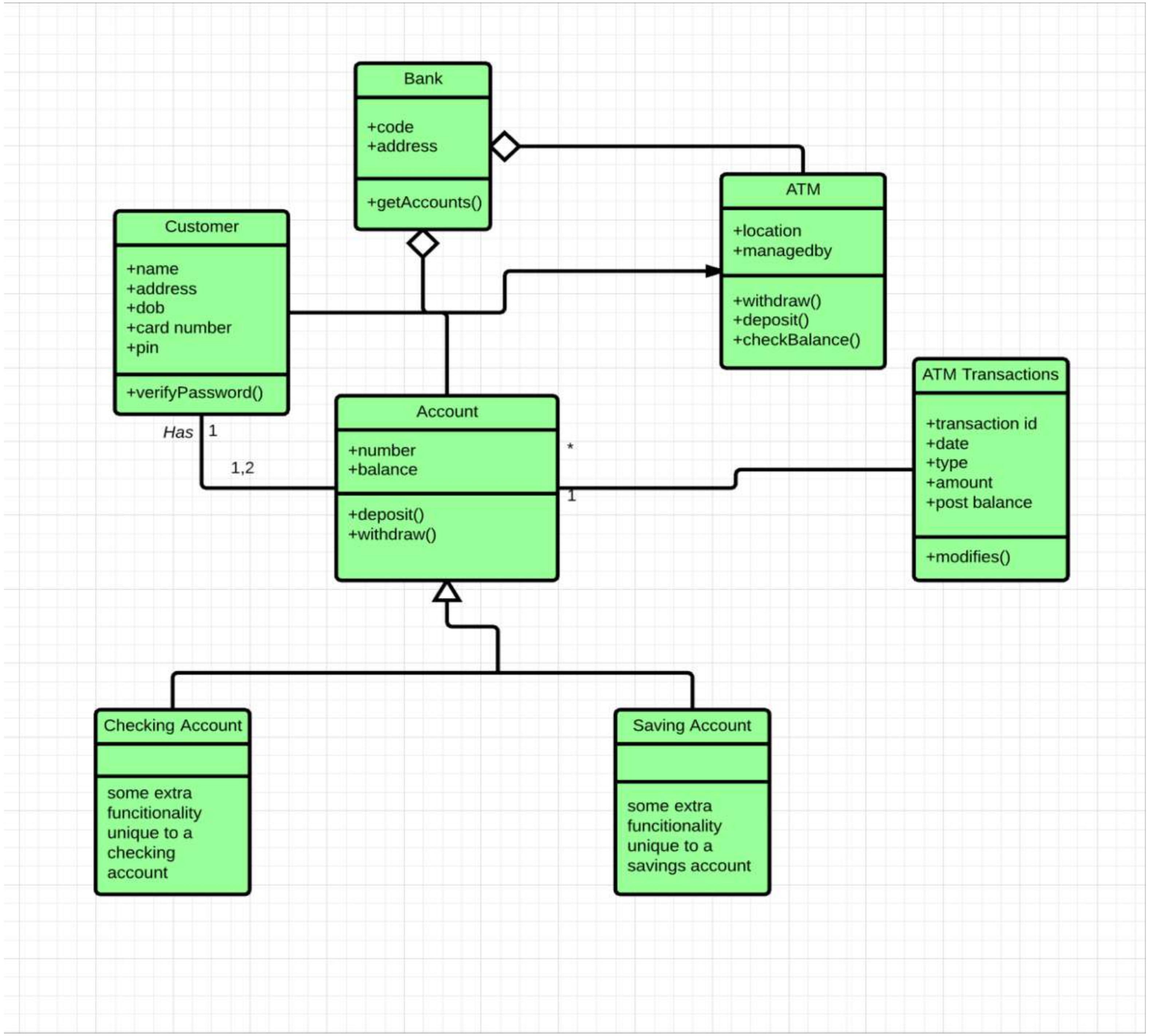
Composition



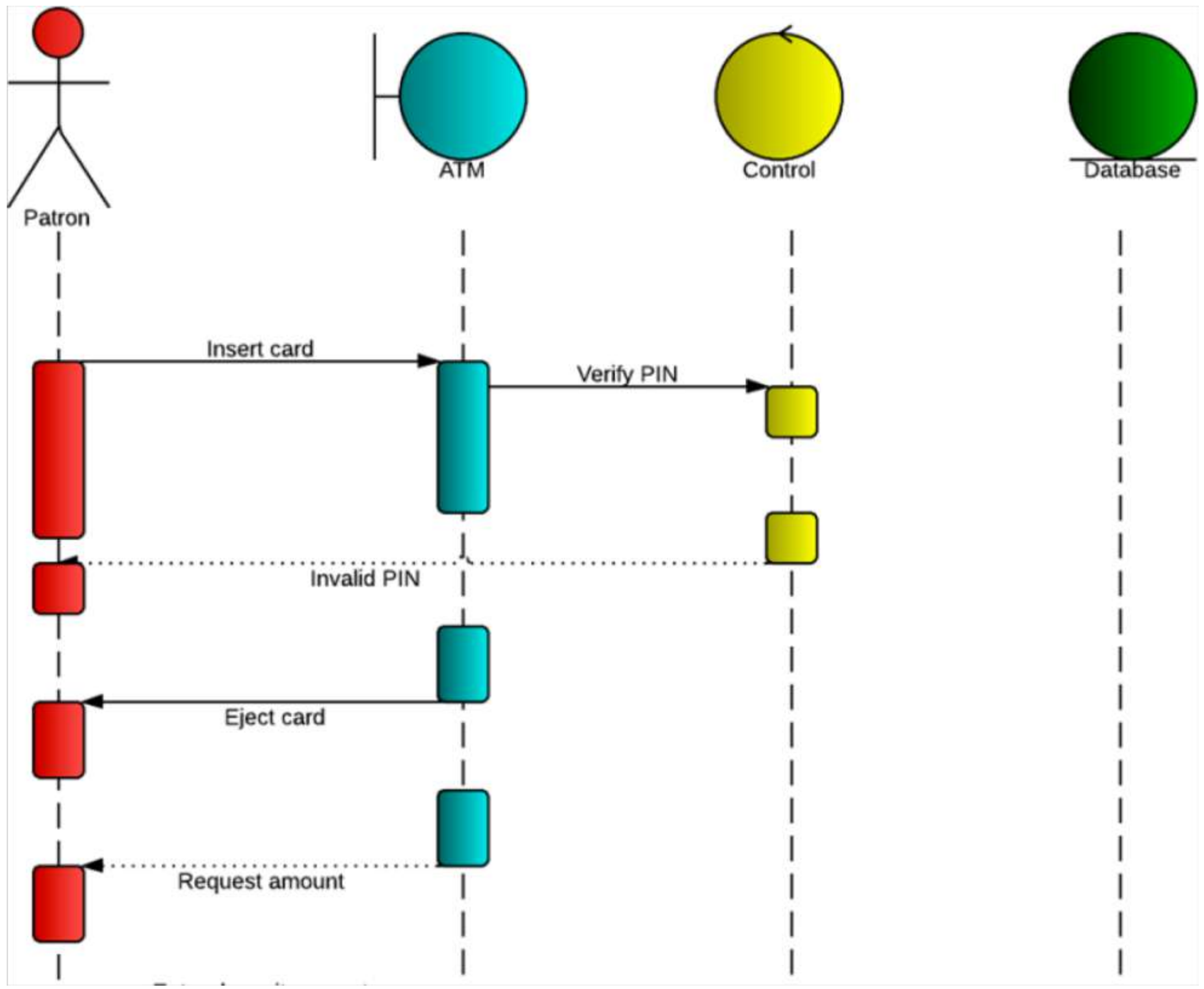
Multiplicity



Structure diagram Notation



Behavior diagrams



That's all Folks!