COMP1023 Software Engineering

Spring Semester 2019-2020

Dr. Radu Muschevici

School of Computer Science, University of Nottingham, Malaysia

Lecture 7

2020-03-25

Topics covered

So far we learnt about **UML Diagrams** for modelling…

System Behaviour

- Use case diagrams (& use case bodies)
- Sequence diagrams
- Activity diagrams

Topics covered

Today we will look at modelling…

System Behaviour

- Use case diagrams (& use case bodies)
- Sequence diagrams
- Activity diagrams

System Structure

- Class diagrams

Modelling System Structure vs. Behaviour

Behavioural models: a dynamic view

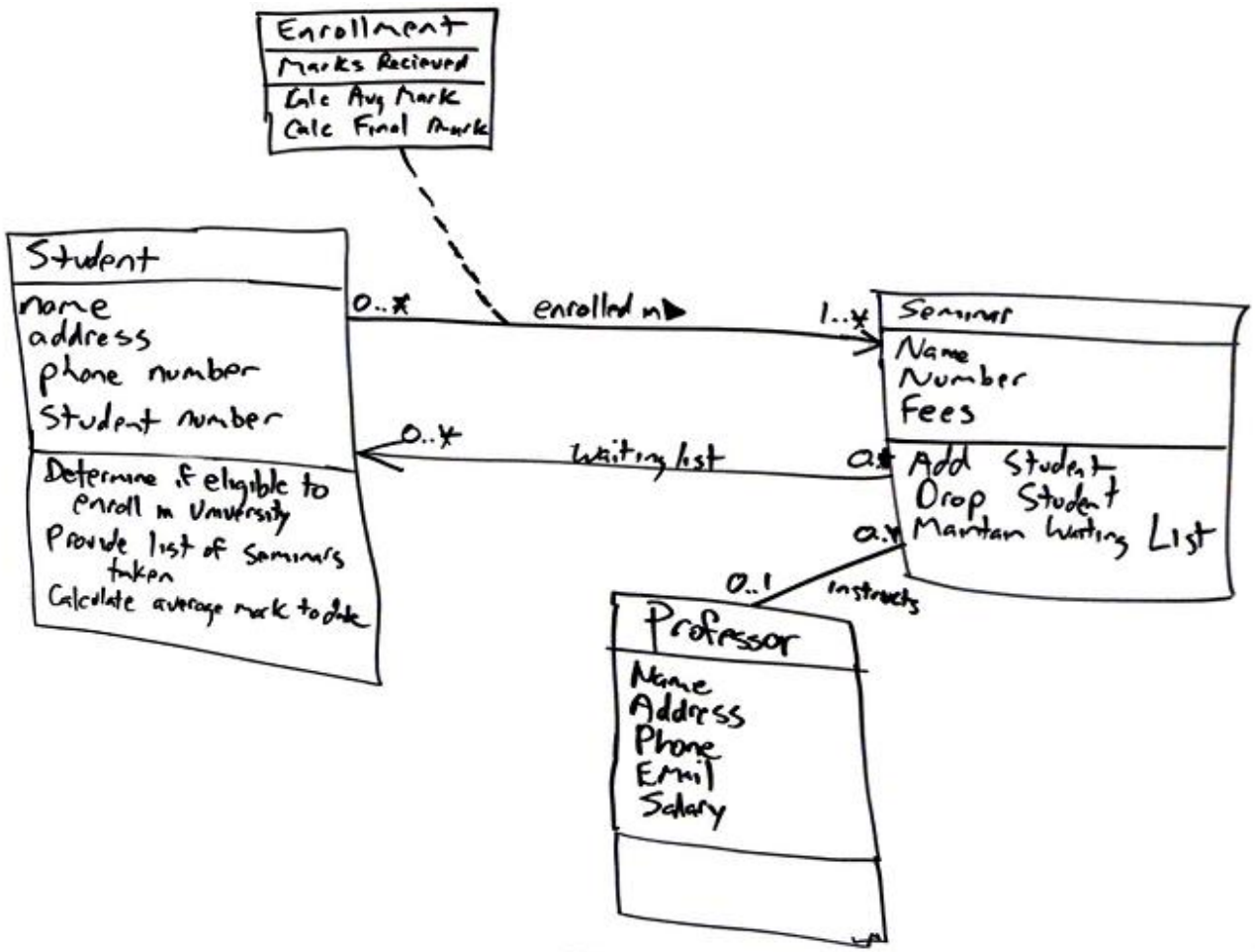
- Emphasize the dynamic (runtime) behavior of the system.
- (Inter)actions and collaborations among objects
- Changes to the internal states of objects
- Typically also model the flow of **time** (time axis, arrows etc.)

Structural models: a static view

- Emphasize the static structure of the system.
- Show system components: classes, objects, attributes, operations
- Show relationships among system components

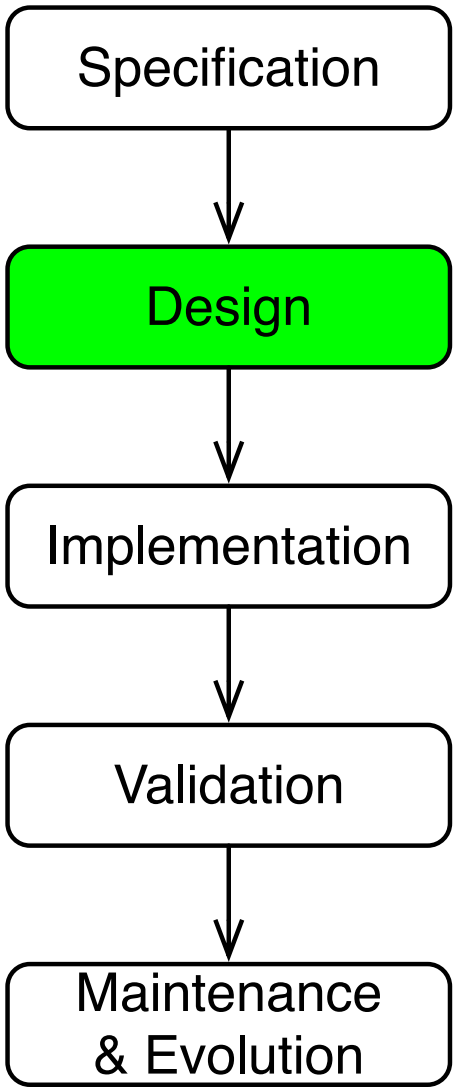
UML Class Diagrams

Used during system **design**, when developing an object-oriented system model to show the **classes** in a system and the **associations** between these classes.



Recap: System Design

- …is one of the basic activities of any software development process (a.k.a. **life cycle**).
- Goal is to develop the overall structure of a solution to the problem.
- Outcomes are individual, buildable **components** and their **relationships** to one another.
- Graphical models used in design phase: UML **sequence**, **activity** & **class** diagrams.



Classes (a quick recap)

- Are the (static) building blocks of **object-oriented** systems (systems that are implemented using an object-oriented programming language).
- Are the **templates** for creating objects (at runtime). Objects are called **instances** of a class.
- Define both the **state** (data) and **behaviour** of objects.
- Can entertain **relationships** to other classes in the system:
  - Hierarchical: generalisation/specialisation (inheritance) — **“is-a”**
  - Compositional: inclusion — **“has-a”**

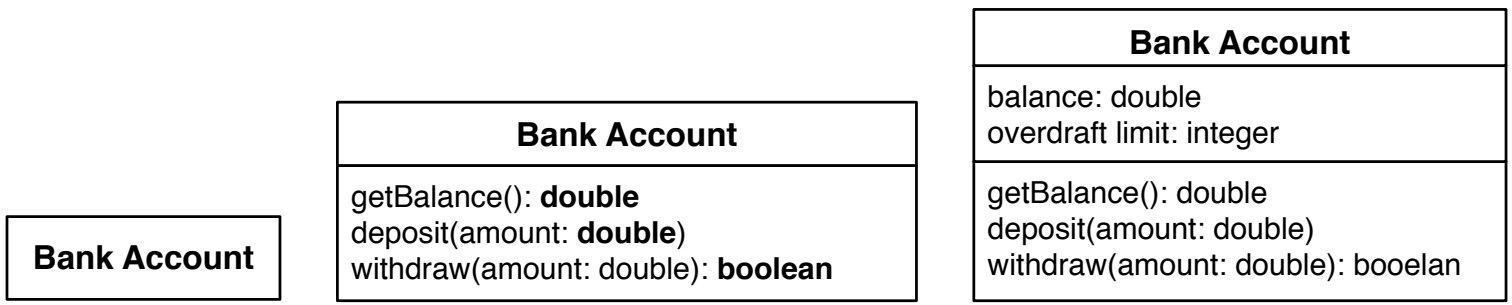
Object-Orientation

- Is a **programming paradigm** (a pattern or model of programming).
- Based on the concept of **objects**: program components that encapsulate **data** (attributes, properties) and exhibit a certain **behaviour** (methods, procedures).
- Most programming languages currently in use are **object-oriented**: Java, C#, Python, Scala, Swift, Kotlin …
- Modern programming languages are increasingly **multi-paradigm**: object-oriented **and** functional **and** concurrent **and** …

Classes

- Are blueprints (templates) for building (**instantiating**) objects.
- A class describes the set of objects that share the same features.
- A class typically has **one responsibility** (accomplishes only one thing).

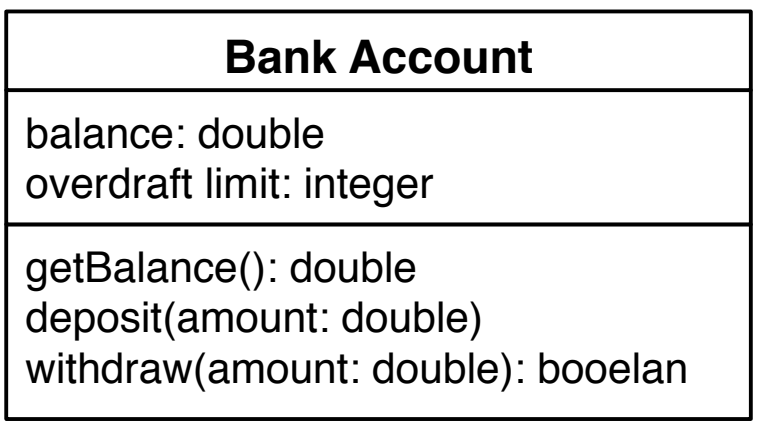
- UML representation (in **class diagrams**): boxes
- Examples:



Classes in UML

A class is typically displayed as a box with three compartments:

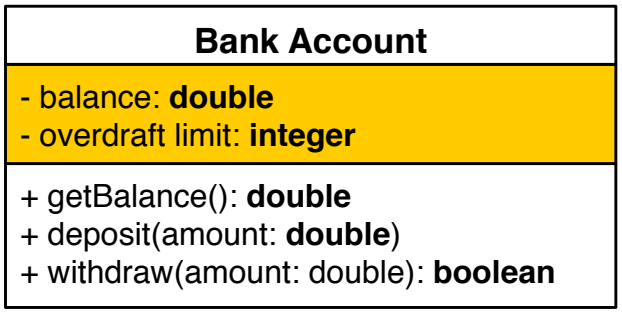
- Name
- Attributes
- Operations



Classses: State

- The class **attributes** (fields) contain the data that each object stores. Together they are the object's **state**.

- In UML, attributes are specified in the middle compartment of the class box.

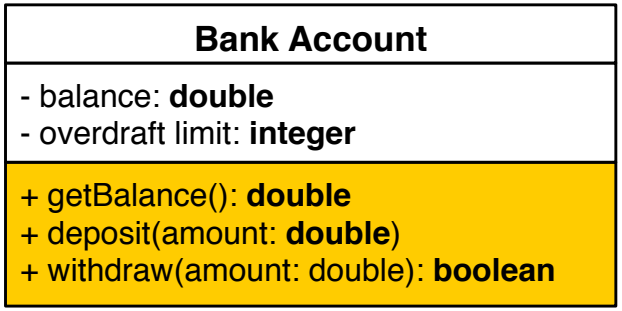


- Attributes typically have:
  - Type
  - Visibility (such as `public[+]`, `private[-]`, `protected[#]` ...) — depending on the programming language used.

Classes: Behaviour

- the class operations (methods) together define the **behaviour** of all objects built from that class.

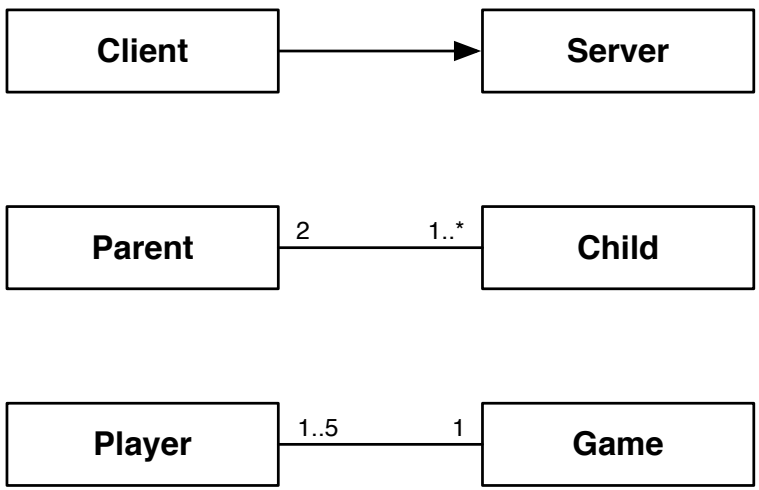
- In UML, the methods are specified in the bottom compartment of the class box.

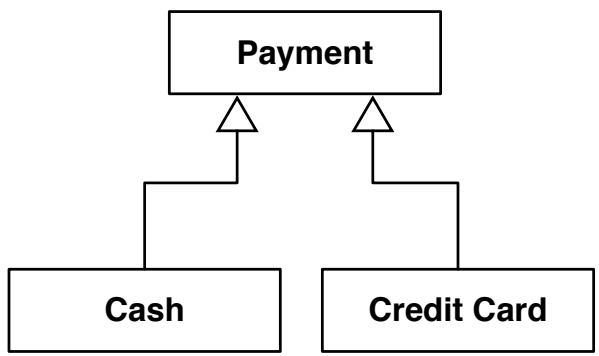


- Operations typically have:
  - Return type
  - Parameters
  - Visibility (such as `public[+]`, `private[-]`, `protected[#]` ...) — depending on the programming language used.

Classes: Relationships

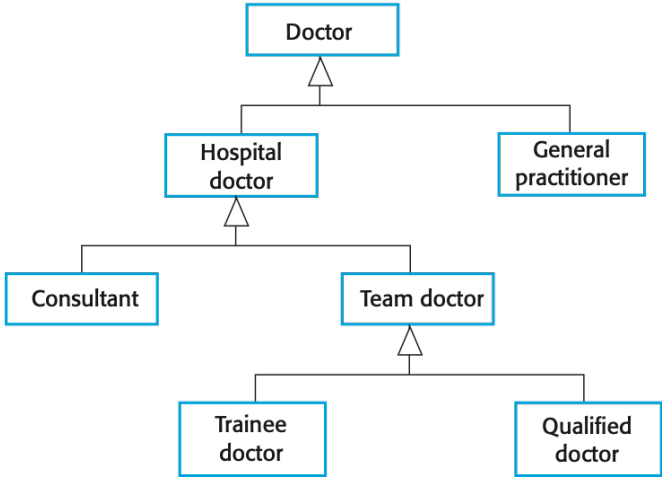
- Relationships are represented as **lines** connecting classes.
- Depending on their type, lines will be adorned with further information.





Generalisation

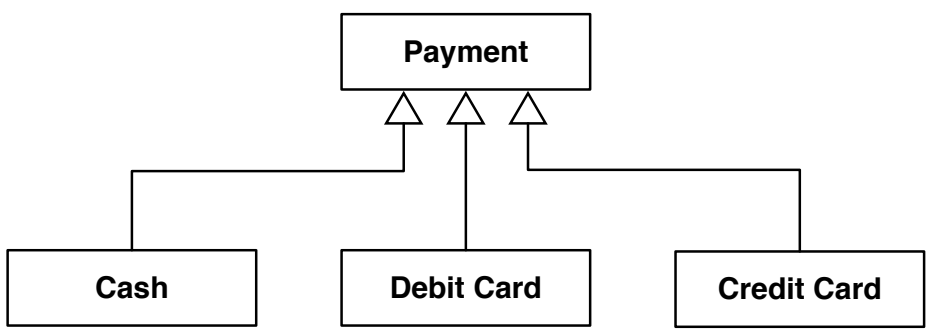
- Generalisation is a everyday technique for managing complexity.
- Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes.
- This allows us to infer that different members of a class have some common characteristics. E.g. squirrels and rats are rodents.
- A generalisation hierarchy:

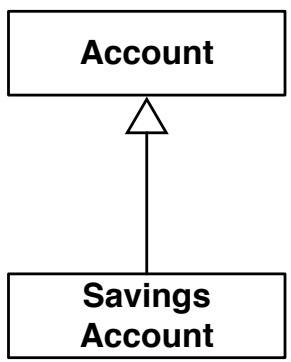


The generalisation relationship (“is-a”)

- A generalization is a relationship between a more general **superclass** and a more specific **subclass**.
- Generalisation is also known as **inheritance**: the subclass incorporates state and behavior from the superclass.
- An instance of the more specific class is **also** an instance of the more general class, so that we can say, e.g. “Savings Account **is an** Account”.

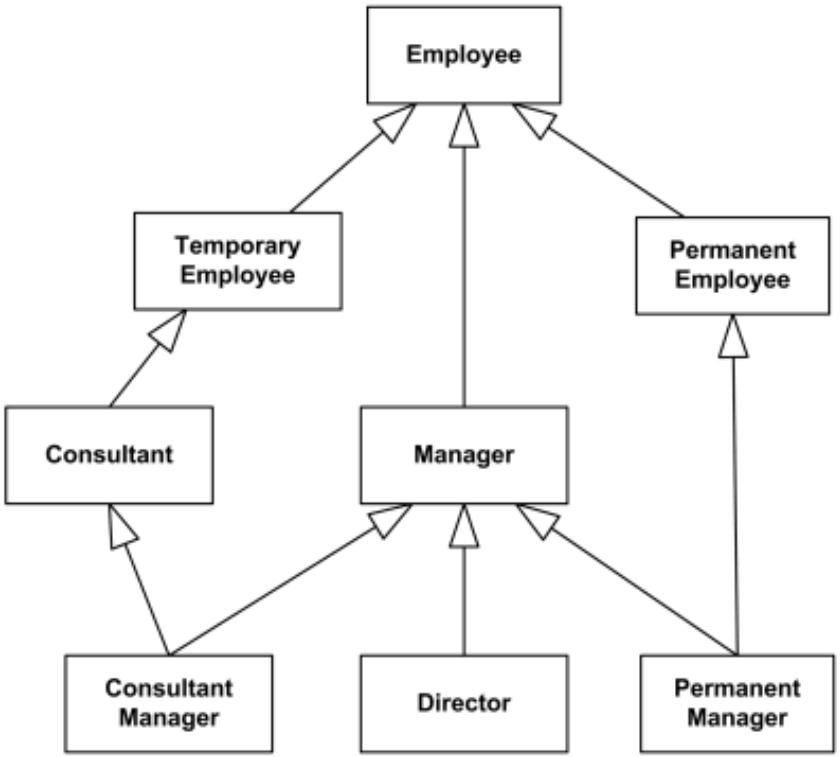
- UML representation using arrow with triangle arrowhead pointing to the superclass:

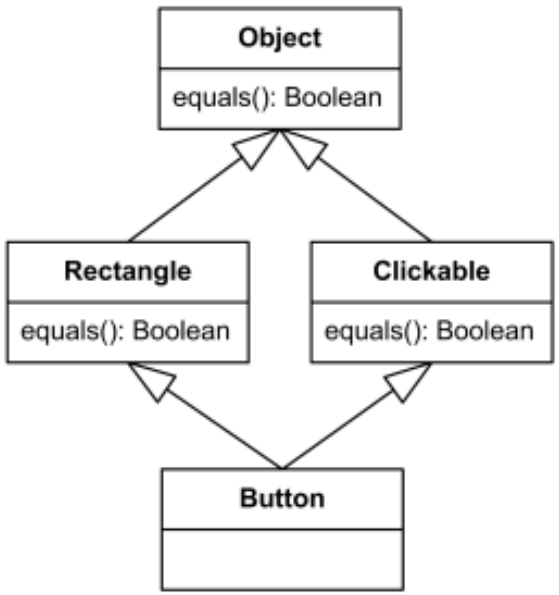




Multiple inheritance

- It is permissible to model inheritance from several superclasses.
- Most languages (Java, C#, …) support only **single inheritance**.
- Multiple inheritance has its issues (see: the “diamond problem”).

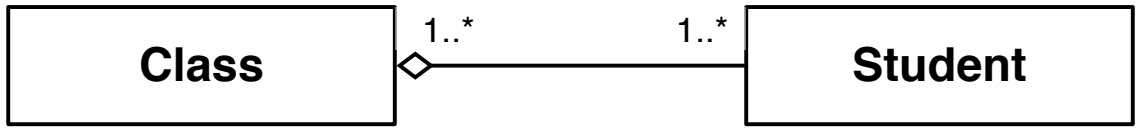




The aggregation relationship

- An aggregation models how classes are composed of other classes, describing a “part-of” or “has-a” relationship.
  - A student is **part of** a class.
  - A class **has a** student (or several students in this case).

- UML representation using line with diamond on the container side. **Multiplicity** indicates the number of elements involved in the relationship on each side.

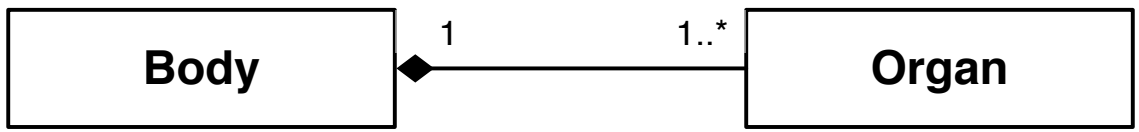


- Aggregation is a “weak” containment relationship: the contained item can exist even if the container does not exist.
  - Student can exist independently of Class.

The composition relationship

- A composition (also) models how classes are composed of other classes, describing a “part-of” or “has-a” relationship.
  - An organ is **part of** a class.
  - A body **has an** organ (or several).

- UML representation using line with **filled** diamond on the container side. **Multiplicity** indicates the number of elements involved in the relationship on each side.



- Composition is a “strong” containment relationship: the contained item cannot exist without the container.
  - Organ cannot exist without the Body that contains it.

Multiplicities

- UML multiplicity allows to specify the **cardinality** (the number of elements) of a collection – for example the number of classes involved in an composition.
- Typically specified with a lower and upper bound (not always).

Examples

Multiplicity	Cardinality of collection
`0`	empty
`0..0`	empty
`0..1`	zero or one instance
`1..*`	one or more instances
`n..m`	at least `n` but no more than `m` instances

Generalization, Aggregation and Composition

Generalisation is a mechanism that allows the data and behavior of one class to be included in or used as the basis for another class.

- A subclass **inherits** all data and behaviour (fields and methods) from its superclass.
- The subclass then defines additional, more specific fields and methods.

Composition/aggregation is the principle of building new abstractions from old (or larger abstractions from smaller).

- A class **contains** another class. A class is a **part** of another class.
- Delegation principle: the whole **delegates** work to its parts.

- **Generalization, aggregation and composition** are mechanisms for sharing (**reusing**) code.

How do we design classes?

Class identification from **requirements specification**

- Nouns are potential classes, class attributes, objects.
- Verbs are potential methods or responsibilities of a class.

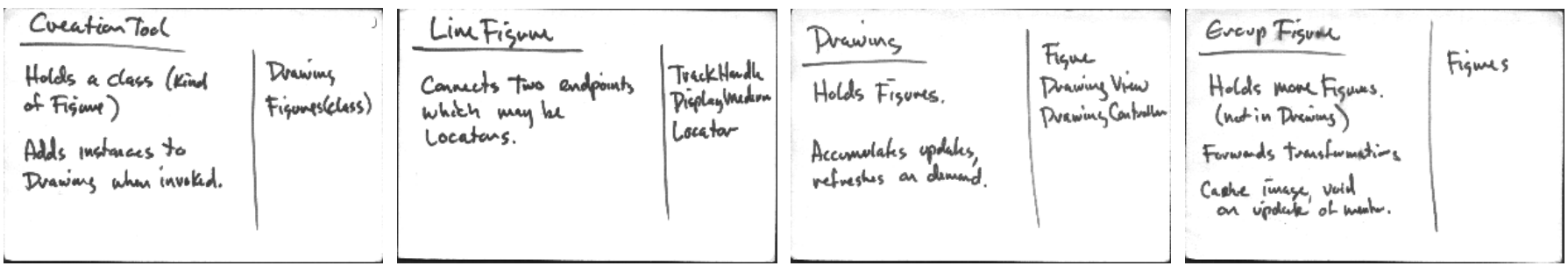
Based on other (already existing) **models**:

- Sequence diagrams
- Activity diagrams
- …

How do we design classes: CRC cards

Class-Responsibility-Collaboration (CRC) cards – for brainstorming

- Typically created from index cards (paper)
- Card is partitioned in three areas:
  1. Name
  2. Responsibilities
  3. Collaborators: other classes with which this class interacts to fulfill its responsibilities



See: https://wiki.c2.com/?CrcCard

Key Points

- UML class diagrams are used in the **design** phase to specify the static structure of object-oriented systems.
- Their level of abstraction can vary from high – for describing the overall component structure of the system – to low – for providing a detailed model ready for **implementation**.
- With the right tool support, code can be generated directly from class diagrams, saving implementation effort (this is known as **model driven engineering**).