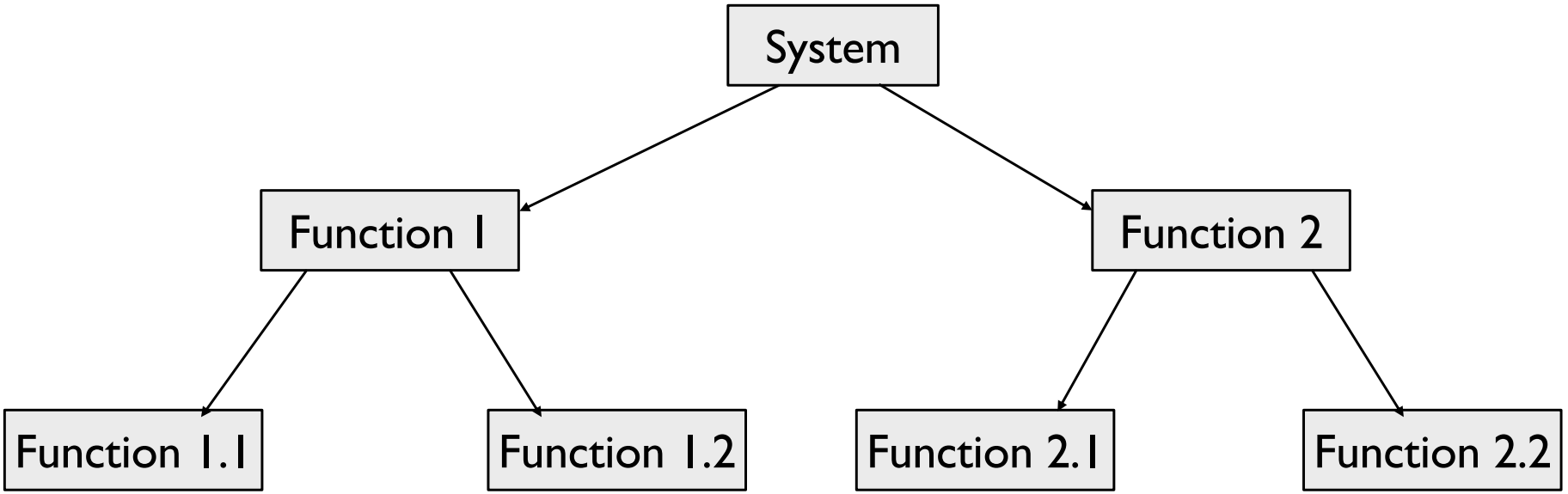
Introduction to object-oriented concepts

- Functional approach
- Object-oriented approach
- Object-oriented concepts
  - Objects
  - Classes
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction

Functional/procedural approach

- Based on specified functions of the system
  - A system consists of several functions
- Decomposition of functions into sub-functions
  - A system consists of sub-systems
  - A sub-system is divided into smaller subsystems



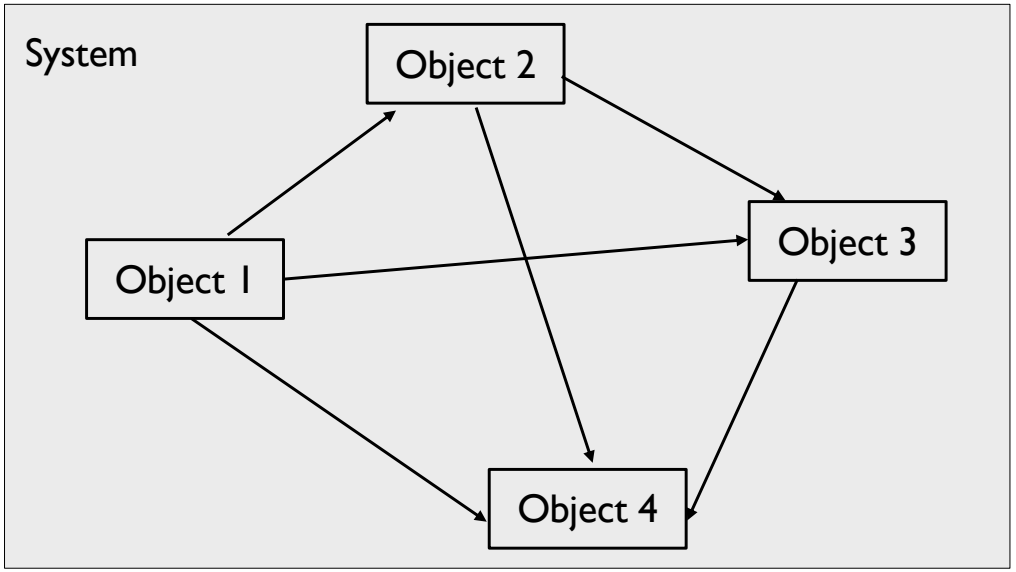
- Functions communicate using shared data or transfer of parameters

Functional approach

- Advantages
  - Easy to apply
  - Work well when data are simple
  - Help to reduce complexity
  - Obtain expected results
- Disadvantages
  - Functions are separated from data
  - Structure of the system is defined based on the functions, therefore a change of functions will cause difficulties in change of the structure
  - The system is weakly open
  - Difficult to re-use
  - An significant maintenance cost

Object-oriented approaches

- The solution of a problem is organized around the concept of objects
- The object is an abstraction of data also containing functions
- A system consists of objects and relationships between them
- Objects communicated by exchanging messages to perform a task
- No global variables
- Encapsulation
- Inheritance



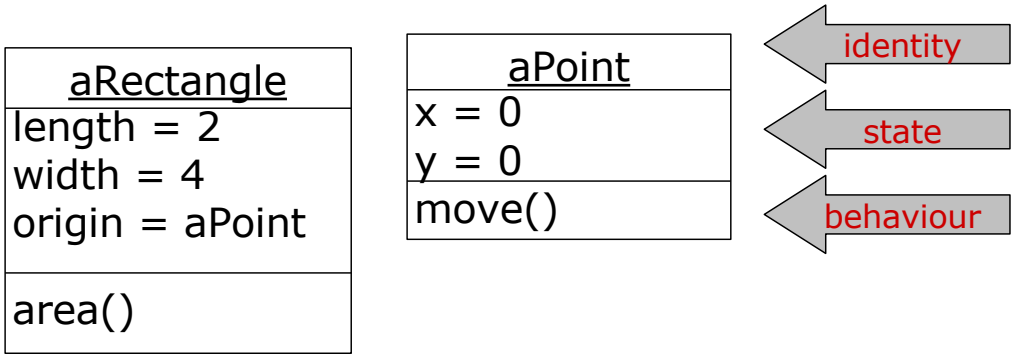
Object-oriented approaches

- Advantages
  - Very close to the real world
  - Easy to reuse
  - Hide information (encapsulation)
  - Lower development cost (inheritance)
  - Suitable for complex systems

- Functional approach v.s. object-oriented approach
  - Functional approach
    - System = algorithms + data structures
  - Object-oriented approaches
    - System = Σ objects
    - Object = algorithms + data structures

Objects

- Object is the concept describing an entity in the real world
- There are relationships between the objects
- Example
  - The Student “Micheal” is an object
  - The Student can’t be an object !
- Object = state + behaviour + identity
  - State (data) describes the characteristics of an object at a given time, and is saved in the variables
  - The behaviour is expressed by the functions of the object
  - Each object has a unique identity
- Example

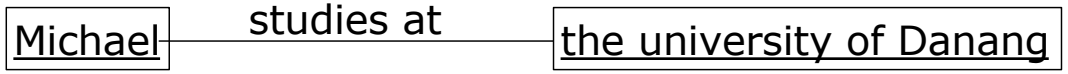


Objects

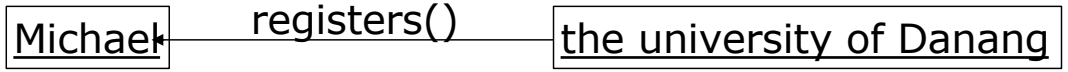
- **State = Set of attributes**
  - An attribute describes one property of the object
  - At every moment, an attribute has a value in a specific set of attributes area
  - Example
    - The car has properties: color, length, width, weight, number of kilometres, …
    - A Renault 207 weighs 1300 pounds, it is red, …
- **Behaviour = Set of functions**
  - A function/method is the ability of the object to perform a task
- The behaviour depends on state
  - Example: A car can start the engine then run, …

Objects

- Links
  - Between objects, there may be links
  - Example



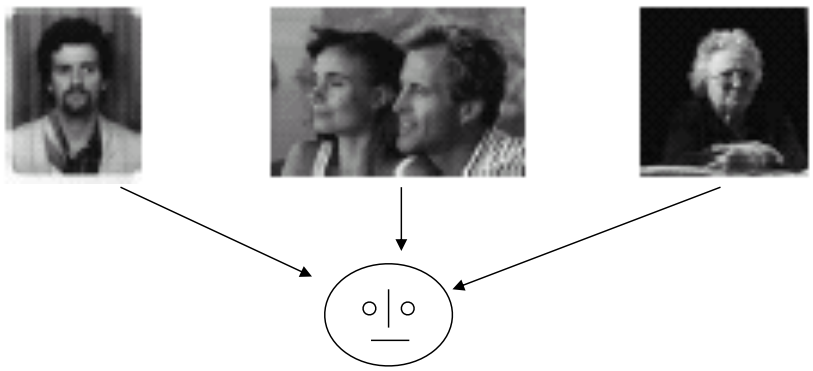
- Communication between objects
  - Send messages



  - Message types
    - constructor
    - destructor
    - getter
    - setter
    - others

Classes

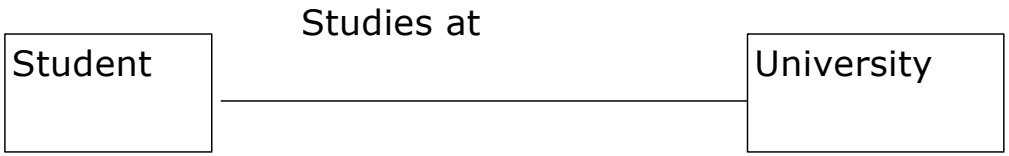
- A class is an abstract description of a set of objects having
  - similar properties
  - common behaviour
  - common relationship with other objects
- Class is an abstraction
  - Abstraction: search for common aspects and omit the differences



  - Reduce the complexity

Class

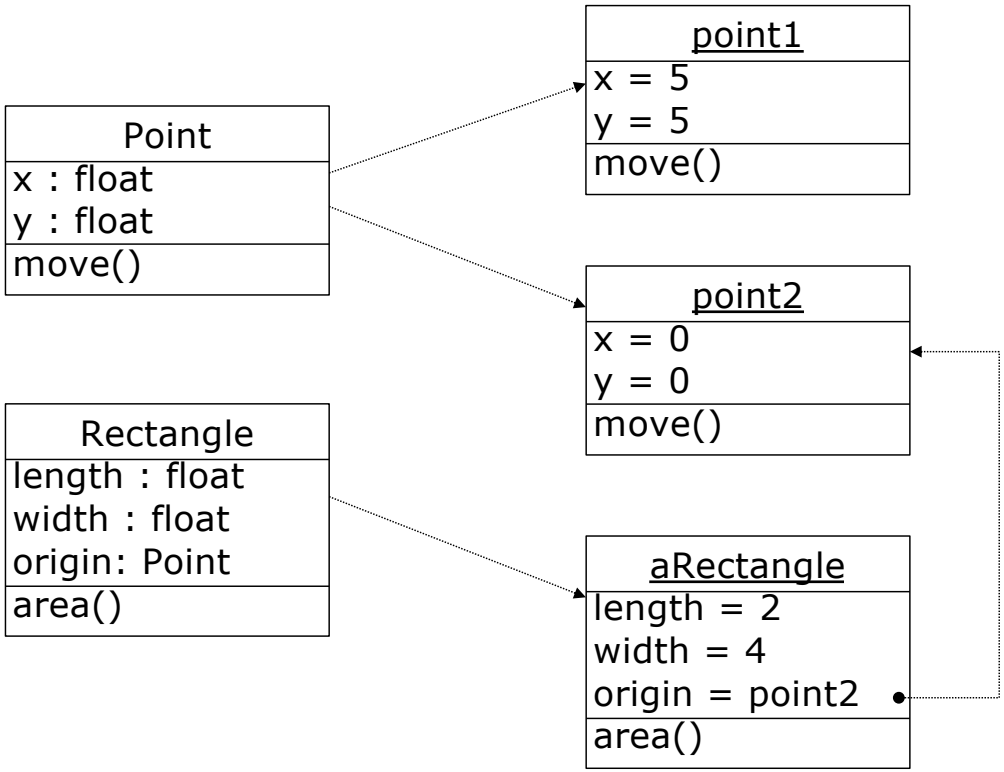
- Relationship
  - There may be relationship between classes
  - A relationship between classes is the set of links between their objects



- Class/Object
  - An object is an instance of a class
  - A value is an instance of an attribute
  - A link between objects is an instance of the relationship between classes

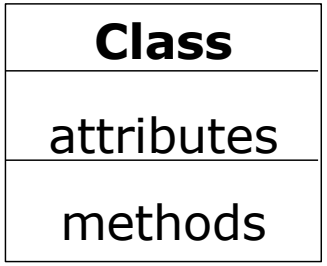
Classes

- Example: Class / Object



Encapsulation

- Data + Processing of data = Object
- Attributes + Methods = Class



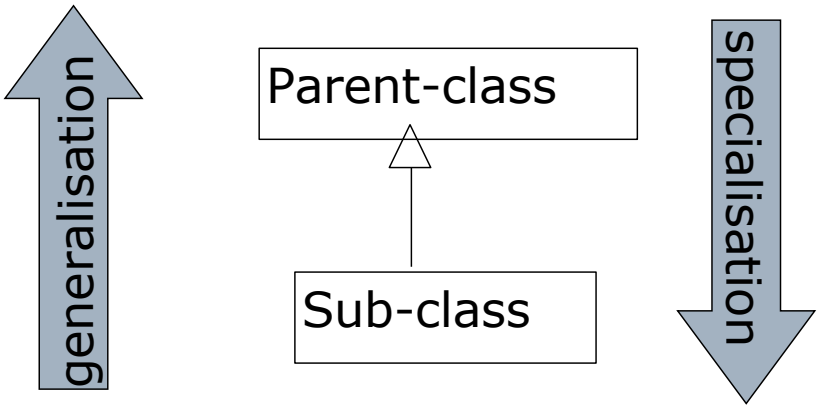
- The state of object is encapsulated by a set of attributes
- The behaviour is encapsulated by a set of methods
  - Users of an object know the messages that the object can receive (public methods)
  - The implementations of methods are hidden from external users

Encapsulation

- Advantages
  - Hide the information
  - Restrict access to the information from the exterior
  - Avoid the global changes in the whole system: the internal implementation can be modified without affecting the external users
  - Facilitate the modularity
  - Easy to reuse
  - Easy to maintain

Inheritance

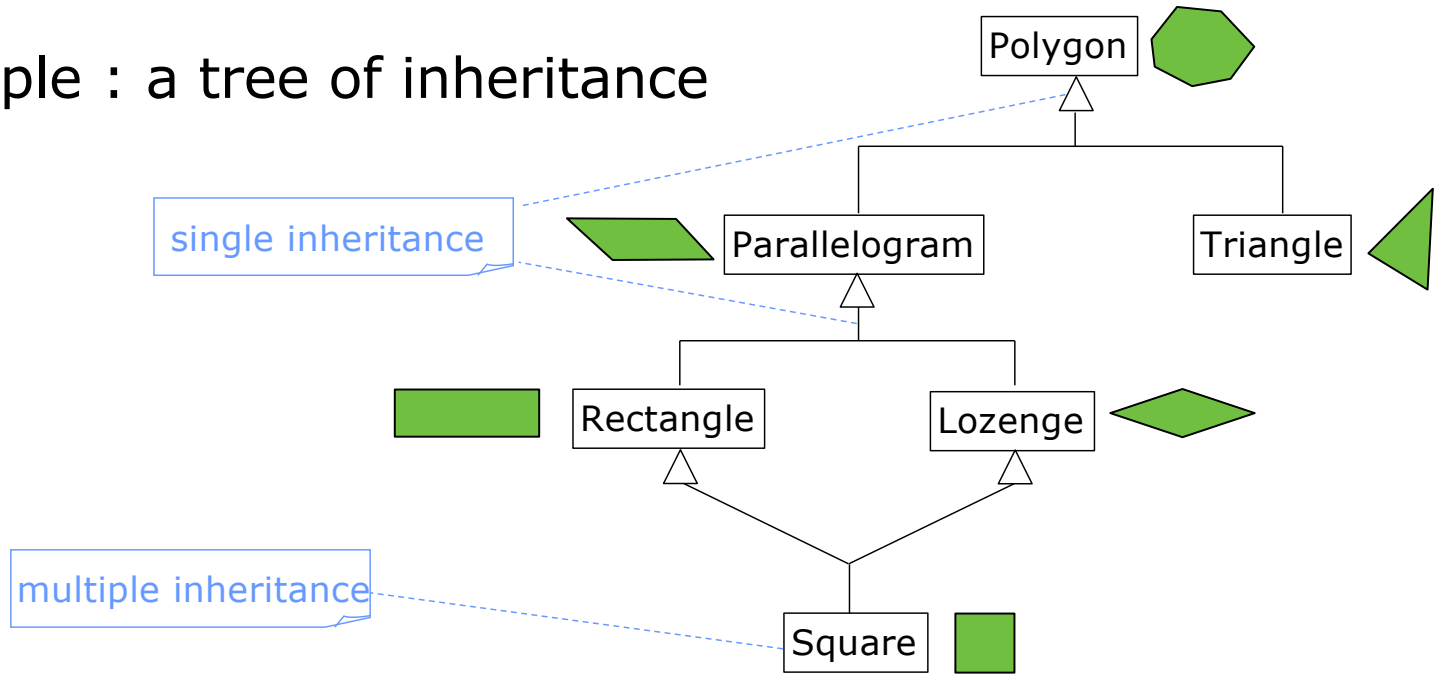
- Inheritance allows the reuse of the state and the behaviour of a class by other classes
- A class is derived from one or more classes by sharing attributes and methods
- Subclass inherits attributes and methods of parent-class
- Generalisation / Specialisation
  - Generalisation: common attributes of sub-classes are used to construct the parent-class
  - Specialisation: sub-classes are constructed from the parent-class by adding other attributes that are unique to them



Inheritance

- **Single inheritance**: a sub-class inherits from only one parent-class
- **Multiple inheritance**: a sub-class inherits from multiple parent-classes

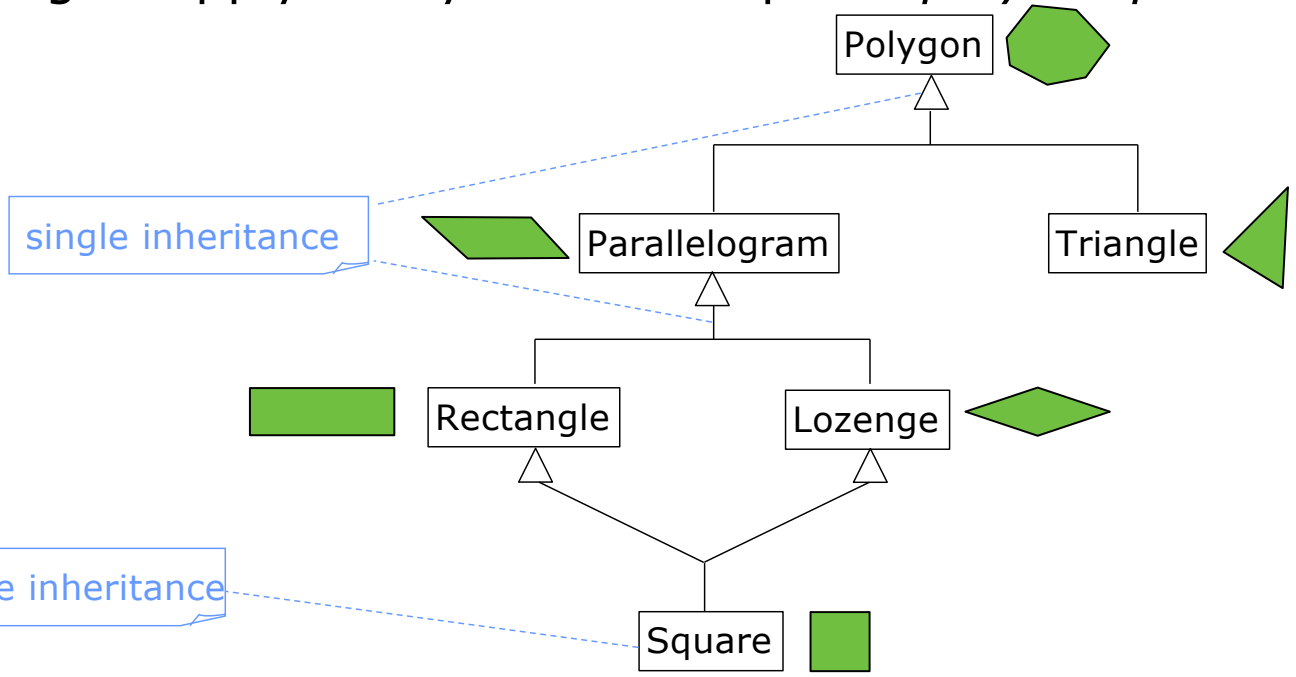
- Example : a tree of inheritance



- What is the difficulty of multiple inheritance?

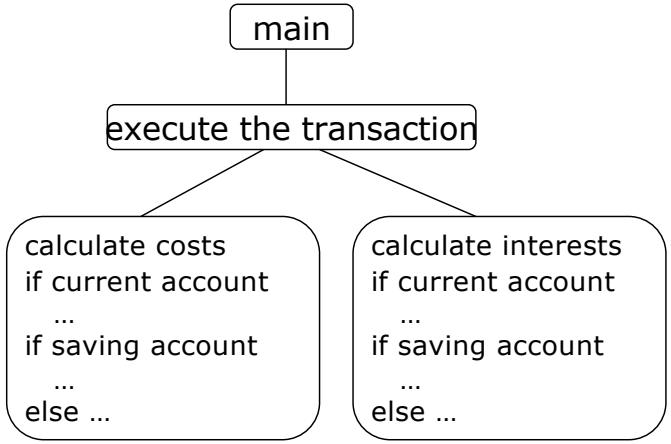
Inheritance

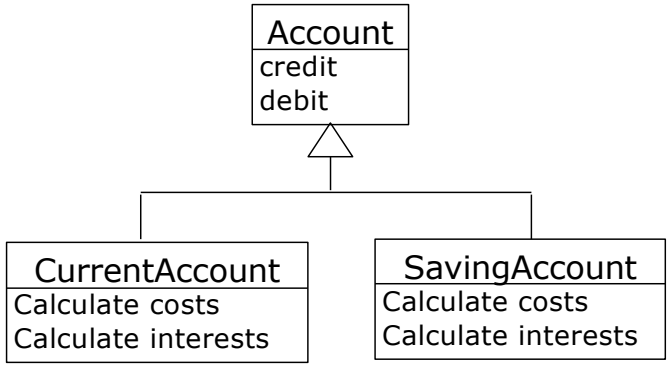
- Advantages
  - Organisation of classes
    - classes are organised hierarchically
    - facilitation of the management of classes
  - Construction of classes
    - sub-classes are constructed from parent-classes
  - Reduction of development cost by avoiding to re-write the code
  - Allowing to apply easily the technique of *polymorphism*



Polymorphism

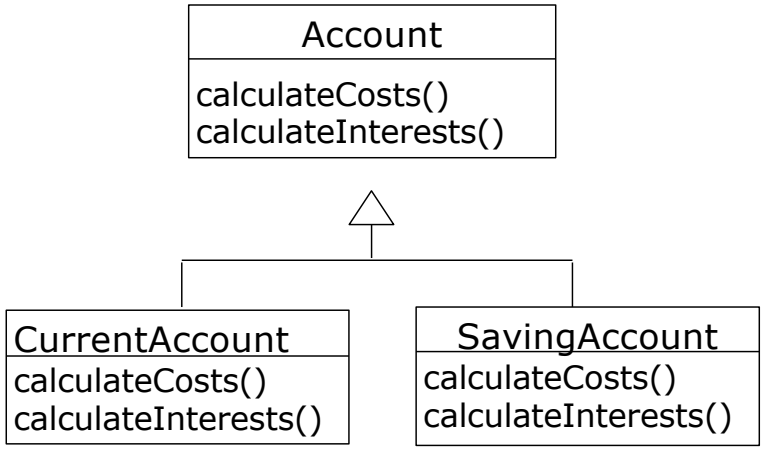
- Polymorphism of methods
  - Different methods are capable of answering to a request
  - Methods having the same name are defined differently (different behaviours) in different classes
  - Sub-classes inherit the specification of methods from parent-class and these methods can be re-defined appropriately
  - Reducing the use of conditional statements (e.g., if-else, switch)
- Procedural approach *versus* Object-oriented approach





Polymorphism: dynamic linking

- The method to be executed by an object depends on the class of the object: dynamic linking
- The dynamic linking is necessary when
  - A variable refers to an object whose class of membership is part of an inheritance tree
  - Several methods exist for the same message (name) in the inheritance tree (polymorphism)

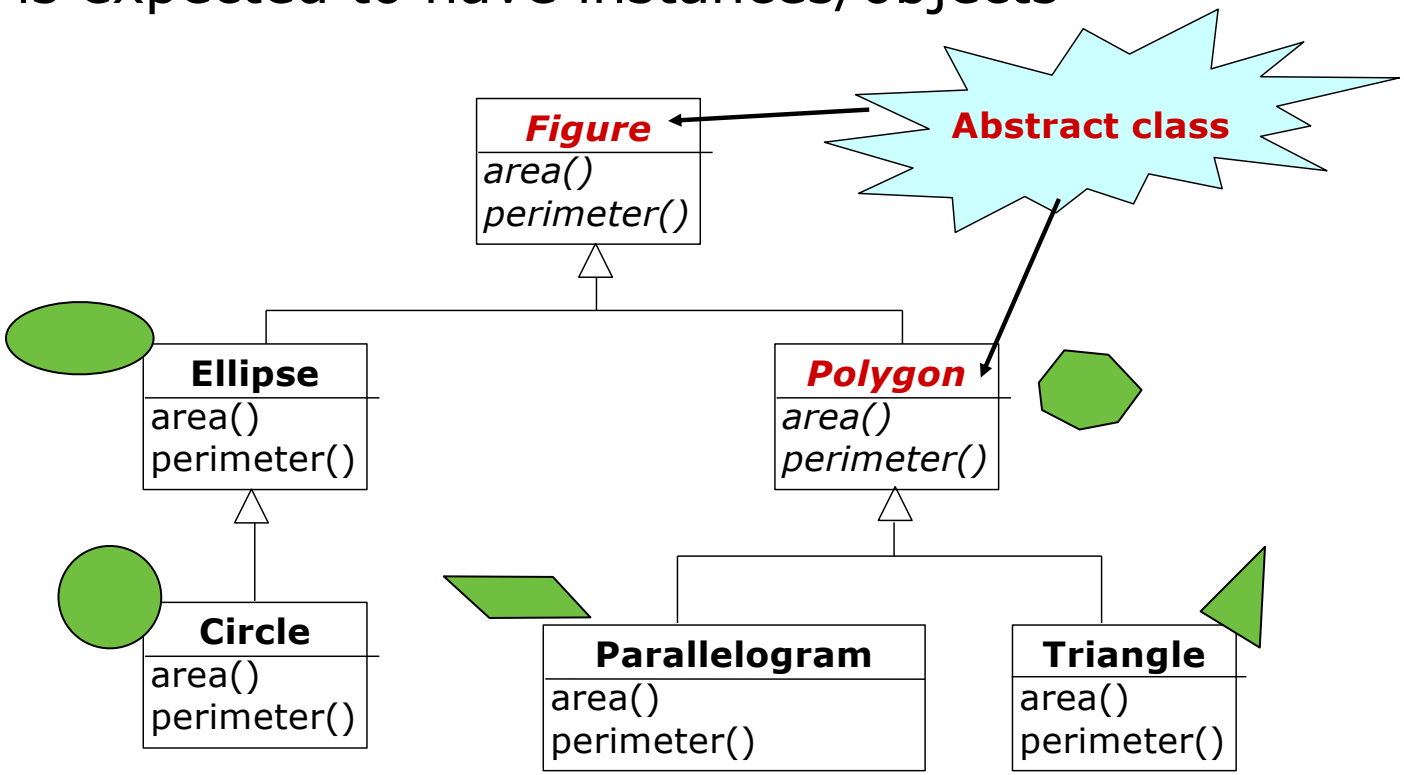



```
int calculateCost(Account accounts)
{
    int s = 0;
    for (int i = 0; i < accounts.length; i++)
        s = s + accounts[i]->calculateCosts();
    return s;
}

void main()
{
    Account accounts = new Account[2];
    accounts[0] = new CurrentAccount();
    accounts[1] = new SavingAccount();
    int s = calculateCost(accounts);
    …
}
```

Abstraction: abstract class

- An abstract class
  - indicates the common characteristics of the sub-classes
  - can’t have instances/objects
- A concrete class
  - contains a complete characterization of real-world objects
  - is expected to have instances/objects



Abstraction: abstract method

- A method should be defined at the highest possible abstraction level
  - At this level, the method can be abstract (i.e., no implementation)
  - In this case, the class is also abstract
  - If a class has an abstract method, at least one of its subclasses must implement this method
    - All the methods of a class at the bottom of the inheritance tree must be concrete