FPT Education

FPT UNIVERSITY

SOFTWARE DESIGN (SWD392)

CH04 – SOFTWARE DESIGN AND ARCHITECTURE CONCEPTS

Main Contents

- Object Oriented Concepts
- Concurrent Processing
- Design Patterns
- Software Architecture & Components
- Software Quality Attributes

Object Oriented Concepts
Objects & Classes 1/2

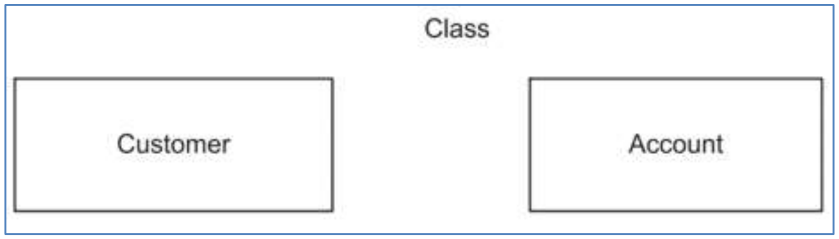
An ***object*** is a real-world physical or conceptual entity that provides an understanding of the real world and, hence, forms the basis for a software solution

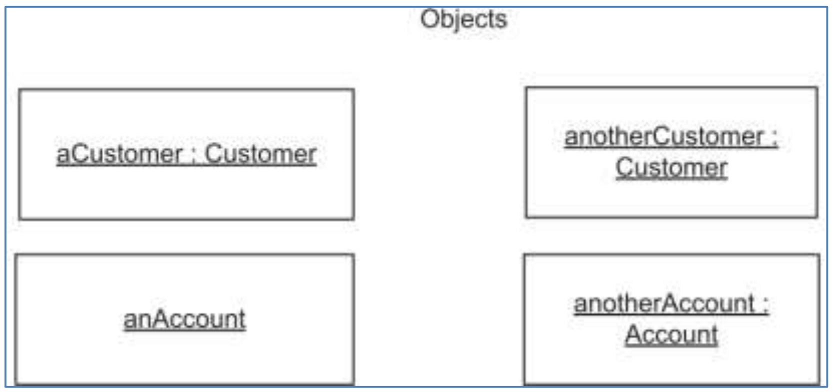
- A real-world object can have physical properties (they can be seen or touched): door, motor, lamp,..
- A conceptual object is a more abstract concept: an account, a transaction,..

An object (*object instance*) is a single “thing”: John’s car or Mary’s account

A ***class*** (object class) is a collection of objects with the same characteristics: Account, Employee, Car, or Customer

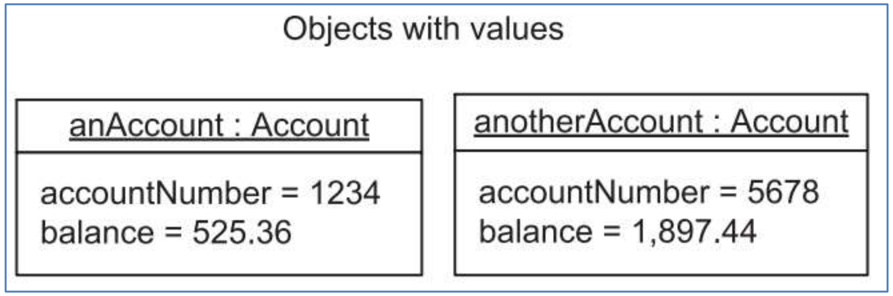
Object-oriented applications consist of objects.

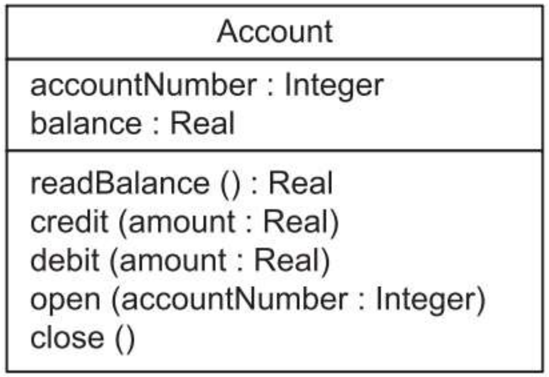




Object Oriented Concepts
Objects & Classes 2/2

- An object groups both data & procedures that operate on the data
  - The procedures are usually called operations or methods.
  - Some approaches, including the UML notation, refer to the operation as the specification of a function performed by an object and the method as the implementation of the function
- An ***attribute*** is a data value held by an object in a class. Each object has a specific value of an attribute.
- An ***operation*** is the specification of a function performed by an object
  - An object has one or more operations.
  - The operations manipulate the values of the attributes maintained by the object.
  - Operations may have input and output parameters.
  - All objects in the same class have the same operations.





FPT Education

FPT UNIVERSITY

Object Oriented Concepts
Information Hiding 1/2

Information hiding is used in designing the object: decide what information should be visible, what should be hidden

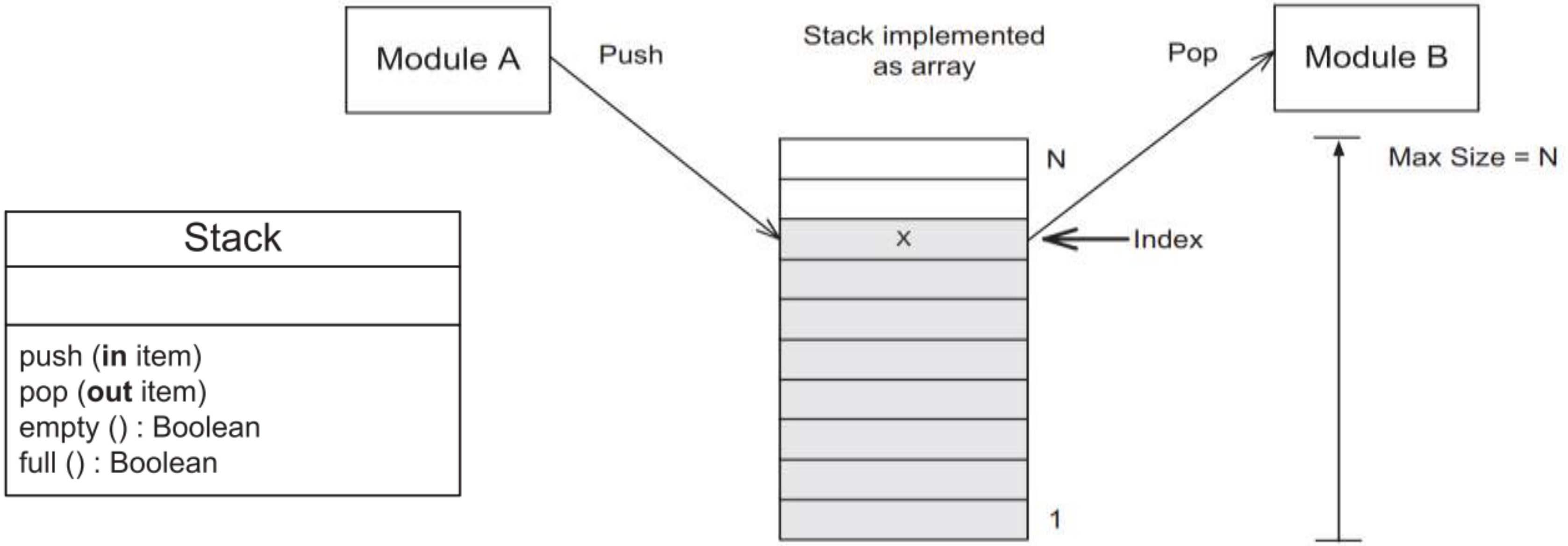
- Hidden parts of an object need not be visible to other objects
- If the internals of the object change -> affect this object only

Encapsulation: the potential change to the hidden information that could potentially change is encapsulated inside an object

- Other objects may only indirectly access the encapsulated data structure by calling the operations of the object.
- The specification of the operations (i.e., the name and the params of the operations) is called the *interface* of the object.

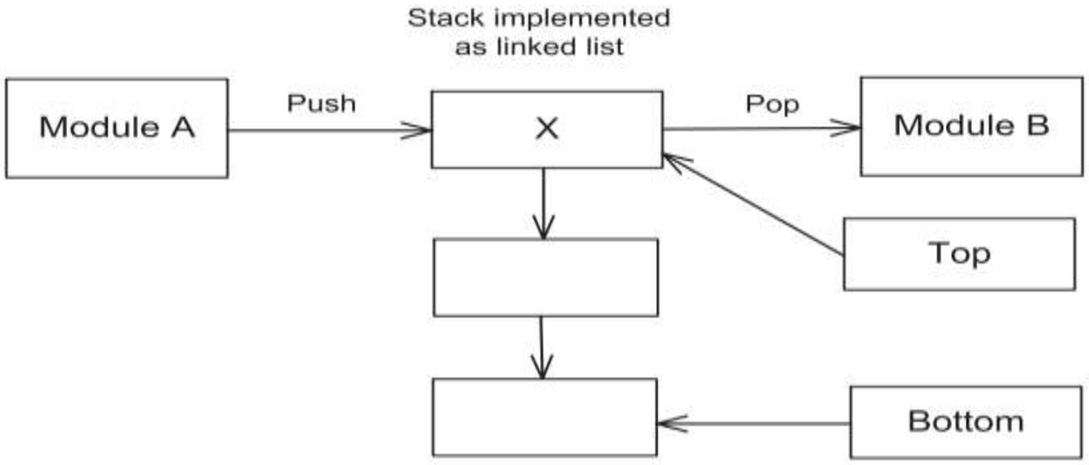
If the data structure changes, the only object affected is the one containing the data structure, not the calling object. This form of information hiding is called ***data abstraction***.

Object Oriented Concepts
Information Hiding 2/2



Stack
push (**in** item) pop (**out** item) empty () : Boolean full () : Boolean

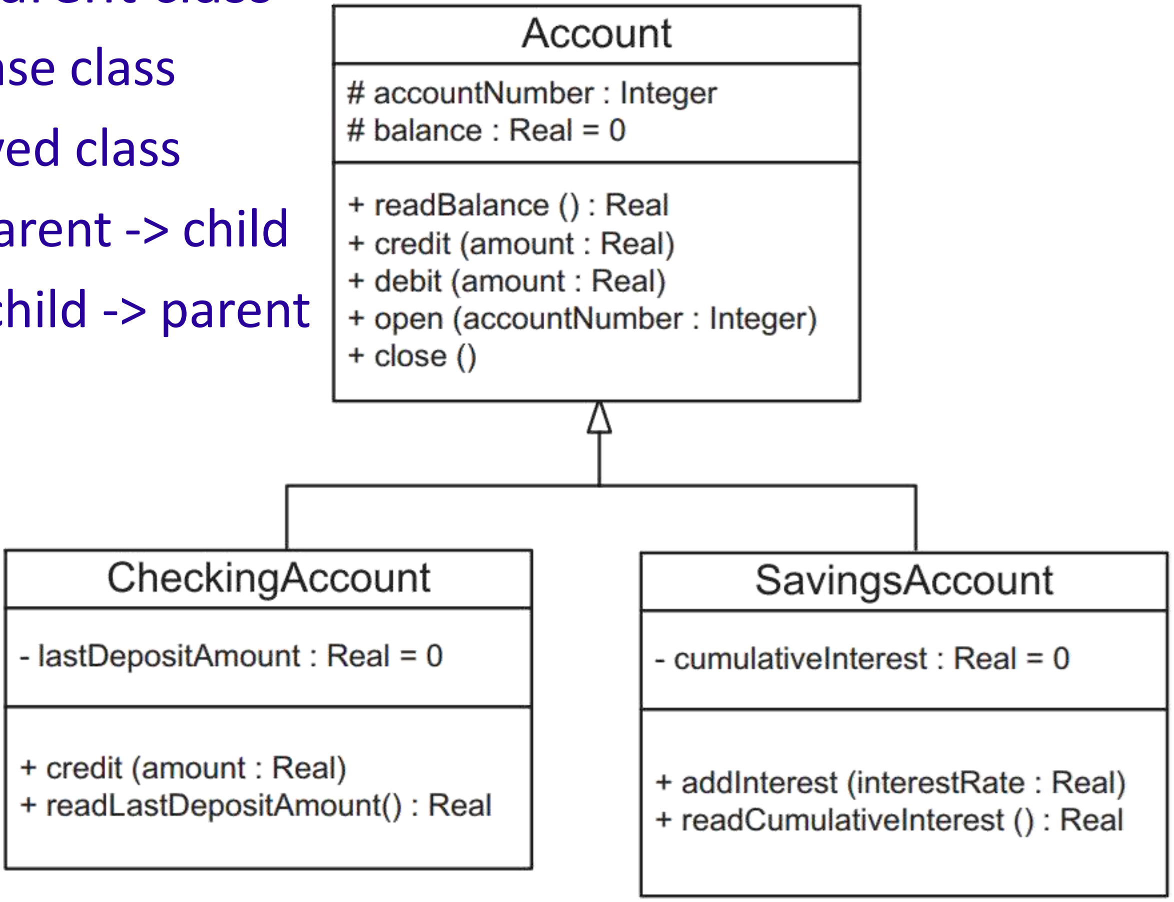
The above is Stack class with a set of operations is defined to manipulate the data structure (array or linked list)



Object Oriented Concepts

Inheritance & Generalization/Specialization

- A mechanism for sharing and reusing code between classes
- A child class inherits the properties (encapsulated data and operations) of a parent class
  - Super class or Base class
  - Subclass or derived class
  - Specialization: parent -> child
  - Generalization: child -> parent


Account
# accountNumber : Integer
# balance : Real = 0
+ readBalance () : Real
+ credit (amount : Real)
+ debit (amount : Real)
+ open (accountNumber : Integer)
+ close ()
CheckingAccount
- lastDepositAmount : Real = 0
+ credit (amount : Real)
+ readLastDepositAmount() : Real
SavingsAccount
- cumulativeInterest : Real = 0
+ addInterest (interestRate : Real)
+ readCumulativeInterest () : Real

Concurrent Processing
Sequential Application

A sequential application is a sequential program that consists of passive objects and has only one thread of control.

- When an object invokes an operation in another object, control is passed from the calling operation to the called operation.
- When the called operation finishes executing, control is passed back to the calling operation.
- In a sequential application, only synchronous message communication (procedure call or method invocation) is supported

Concurrent Processing
Concurrent Application 1/2

In a concurrent application, there are typically several concurrent objects, each with its own thread of control.

- A concurrent source object can send an asynchronous message to a concurrent destination object and then continue executing, regardless of when the destination object receives the message.
- If the destination object is busy when the message arrives, the message is buffered for the object.

Concurrent Processing
Concurrent Application 2/2

In a concurrent application, there are typically several concurrent objects, each with its own thread of control.

- Concurrent source object can send asynchronous message(s) to a concurrent destination object and then continue executing, regardless of when the destination object receives the message.
- If the destination object is busy when the message arrives, the message is buffered for the object.

Concurrent Processing

Concurrent Objects

Also referred to as active objects, concurrent processes, concurrent tasks, or threads

- They have their own thread of control
- Execute independently of other objects.
- They are different from passive objects (invoked)
- No concurrency is allowed within a concurrent object

Concurrent objects often execute asynchronously and are relatively independent of each other for significant periods of time.

Concurrent Processing

Cooperation between Concurrent Objects

Common arise problems in concurrent processing

- The ***mutual exclusion problem*** occurs when concurrent objects need to have exclusive access to a resource, such as shared data or a physical device.
- The ***synchronization problem*** occurs when two concurrent objects need to synchronize their operations with each other.
- The ***producer/consumer problem*** occurs when concurrent objects need to communicate with each other in order to pass data from one concurrent object to another (Inter Process Communication - IPC)

Design Patterns

- Describes a recurring design problem to be solved, a solution to the problem, and the context in which that solution works (***microarchitecture***)
- The main kinds of reusable patterns are as follows:
  - ***Design patterns***: a small group of collaborating objects
  - ***Architectural patterns***: larger-grained (higher level) than design patterns, structure of major subsystems of a system
  - ***Analysis patterns***: recurring patterns found in object-oriented analysis and described them with static models, expressed in class diagrams
  - ***Product line–specific patterns***: concentrating on a specific application domain, provide more tailored domain-specific solutions
  - ***Idioms***: low-level patterns that are specific to a given programming language and describe implementation solutions to a problem that use the features of the language (Java, C++,…)

Software Architecture & Components

A software architecture separates the overall structure of the system, in terms of components and their interconnections, from the internal details of the individual components

- Components: the system modules that could be developed in different ways depending on the particular platform the software architecture.
- To fully specify a component, it is necessary to define it in terms of the operations it ***provides*** and the operations it ***requires***.
- ***Connectors:***
  - Join the components,
  - Encapsulates the interconnection protocol between two or more components: asynchronous (loosely coupled) or synchronous (tightly coupled)

FPT Education
FPT UNIVERSITY

Software Quality Attributes

- Quality requirement of the software, often referred to as nonfunctional requirements
  - Security: system is resistant to security threats
  - Modifiability: modified during or after initial development
  - Reusability: software is capable of being reused
  - Testability: capable of being tested
  - Performance: performance goals (throughput, response times)
  - Availability: capable of addressing, recovering from system failure
  - Maintainability: capable of being changed after deployment
  - Scalability: capable growing after its initial deployment
  - Traceability: product of each phase can be traced back to products of previous phases
- The quality attributes are addressed and evaluated at the time the software architecture is developed, and can have a profound effect on the quality of a software product

Q
A