SS ZG653 (RL 7.1): Software Architecture

Introduction to OODesign

Instructor: Prof. Santonu Sarkar

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Online shopping application- Pet Store

by Sun Microsystems (Oracle)

- Storefront has the main user interface in a Web front-end. Customers use the Storefront to place orders for pets
  - Register User
  - Login user
  - Browse catalog of products
  - Place order to OPC (asynchronous messaging)
- Order Processing Center (OPC) receives orders from the Storefront.
- Administrators
  - Examine pending orders
  - Approve or deny a pending order
- Supplier
  - View and edit the inventory
  - Fulfills orders from the OPC from inventory and invoices the OPC.

System quality

- Availability
- Modifiability
- Performance
- Security
- Testability
- Usability
- Interoperability

- Show orders in multiple languages
- Help in browsing products
- Easy checkout
- Assurance to customer, supplier and bank
- Guaranteed delivery
- Shopping cart is only modified by the customer
- Order never fails
- System is always ready to sell
- Order is always processed in 2sec.
- System always optimally use the hardware infrastructure and performs load-balancing
- Uses standard protocol to communicate with Suppliers
- Uses secure electronic transaction protocol (SET) for credit card processing
- Quite easy to add new supplier
- Little change necessary to add a third party vendor
- Quite easy to sell books in addition to Pets
- Logging is extensive to trace the root cause of the fault

Example

Programming for this Application

- Everything is an object
  - Pet, Customer, Supplier, Credit card, Customer's address, order processing center
- Pets can be sold, credit card can be charged, clients can be authenticated, order can be fulfilled!
  - Have behavior, properties
- Interact with each other
  - Storefront places order to OPC

What's cool?

- Very close to the problem domain… domain experts can pitch in easily.
- Could group features and related operations together
- It's possible to add new types of Pets (e.g., add Birds in addition to Cat, Dog and Fish)

What is a class

- Class represents an abstract, user-defined type
  - Structure – definition of properties/attributes
    - Commonly known as member variables
  - Behavior – operation specification
    - Set of methods or member functions

- An object is an instance of a class. It can be instantiated (or created) w/o a class

Object State

- **Properties/Attribute Values at a particular moment represents the state of an object**
- **Object may or may not change state in response to an outside stimuli.**
- **Objects whose state can not be altered after creation are known as immutable objects and class as immutable class [String class in Java]**
- **Objects whose state can be altered after creation are known as mutable objects and class as mutable class [StringBuffer class in Java]**

States of Three Different INSTANCES of "Dog" class

Labrador	***Golden Retriever***	***Pug***
Age: 1 yr	***Age: 6months***	***Age: 1.5yr***
Price : 3500	***Price : 2000***	***Price : 1500***
Sex:Male	***Sex:Female***	***Sex: Female***

Object-Oriented Programming

- A program is a bunch of objects telling each other what to do, by sending messages
  - In Java, C++, C# one object (say o1) invokes a method of another object (o2), which performs operations on o2
  - You can create any type of objects you want
- OO different from procedural?
  - No difference at all: Every reasonable language is ultimate the same!!
  - Very different if you want to build a large system
    - Increases understandability
    - Less chance of committing errors
    - Makes modifications faster
    - Compilers can perform stronger error detections

Type System

- Classes are user-defined data-types
- Primitive types
  - int, float, char, boolean in Java (bool in C#), double, short, long, string in C#
- Unified type
  - C# keyword object – mother of all types (root)
    - Everything including primitive types are objects
  - Java JDK gives java.lang.Object– not a part of the language
    - Primitive types are not objects

References and Values

Value

- Primitive types are accessed by value
- C++ allow a variable to have object as its value
- C# uses struct to define types whose variables are values
- No explicit object creation/deletion required
  - Faster, space decided during compilation

Reference

- Java allows a variable to have reference to an object only
- C++ uses pointers for references
- Needs explicit object creation
  - Slower, space allocated during runtime from heap
- Java performs escape analysis for faster allocation

Modules

- You need an organization when you deal with large body of software – 20MLOC, 30000 classes or files!
- Notion of module introduced in 1970
  - Group similar functionality together
  - Hide implementation and expose interface
  - Earlier languages like Modula, Ada introduced this notion
  - In OO language “class” was synonymous to a module
- But they all faced the problem of managing 20M lines of C or C++ code
- ANSI C++, Java, Haskell, C#, Perl, Python, PHP all support modules
  - Namespace (C++, C#)
  - Package (Java, Perl)

Module- aka namespace, package

- A set of classes grouped into a module
  - A module is decomposed into sub-modules
  - Containment hierarchy of modules – forms a tree
- A fully qualified, unique name= module+name of the class
  - namespace
  - package
- Import- A class can selectively use one or more classes in a module or import the entire module

Inheritance

- Parent (called Base class) and children classes (Derived classes)
  - A Derived class inherits the methods and member variables of the Base -- also called ISA relationship
  - A child can have multiple parents – Multiple inheritance
  - Hierarchy of inheritance (DAG)

- The Derived class can
  - Use the inherited variables and methods – reuse
  - Add new methods – extends the functionality
  - Modify derived method- called **overriding** the base class member function

Introducing Interface

What is it?

- A published declaration of a set of services
  - An interface is a collection of constants and method declarations
  - No implementation, a separate class needs to implement an interface
- A class can implement more than one interfaces

Why?

- Provide capability for unrelated classes to implement a set of common methods
- Standardize interaction
- Extension- let's the designer to defer the design
- User does not know who implemented it
  - It is easy to change implementation without impacting the user

Interface Definition

- An interface in C++, C# is a class that has at least one pure virtual method
  - A pure virtual method only has specification, but no body
  - This is called abstract base class
  - One can have an abstract class where some methods are concrete (with implementation) and some as pure virtual which could be clumsy

- Java uses keyword “interface” and is more clean
  - Interface only provides method declarations

- Java also allows to define an abstract base class just like C++

Creating Objects

- With built-in types like **int** or **char**, we just say

  int i; char c; --- and we get them

- When we define a class A-- user-defined type
  - We need to explicitly
    1. tell that we want a new object of type A (operator new)
    2. Initialize the object (you need to decide) after creation -- constructor method
       - Special method that compiler understands. The constructor method name must be same as the class name
       - C++ allows constructor method for struct also
- Constructor methods can be overloaded

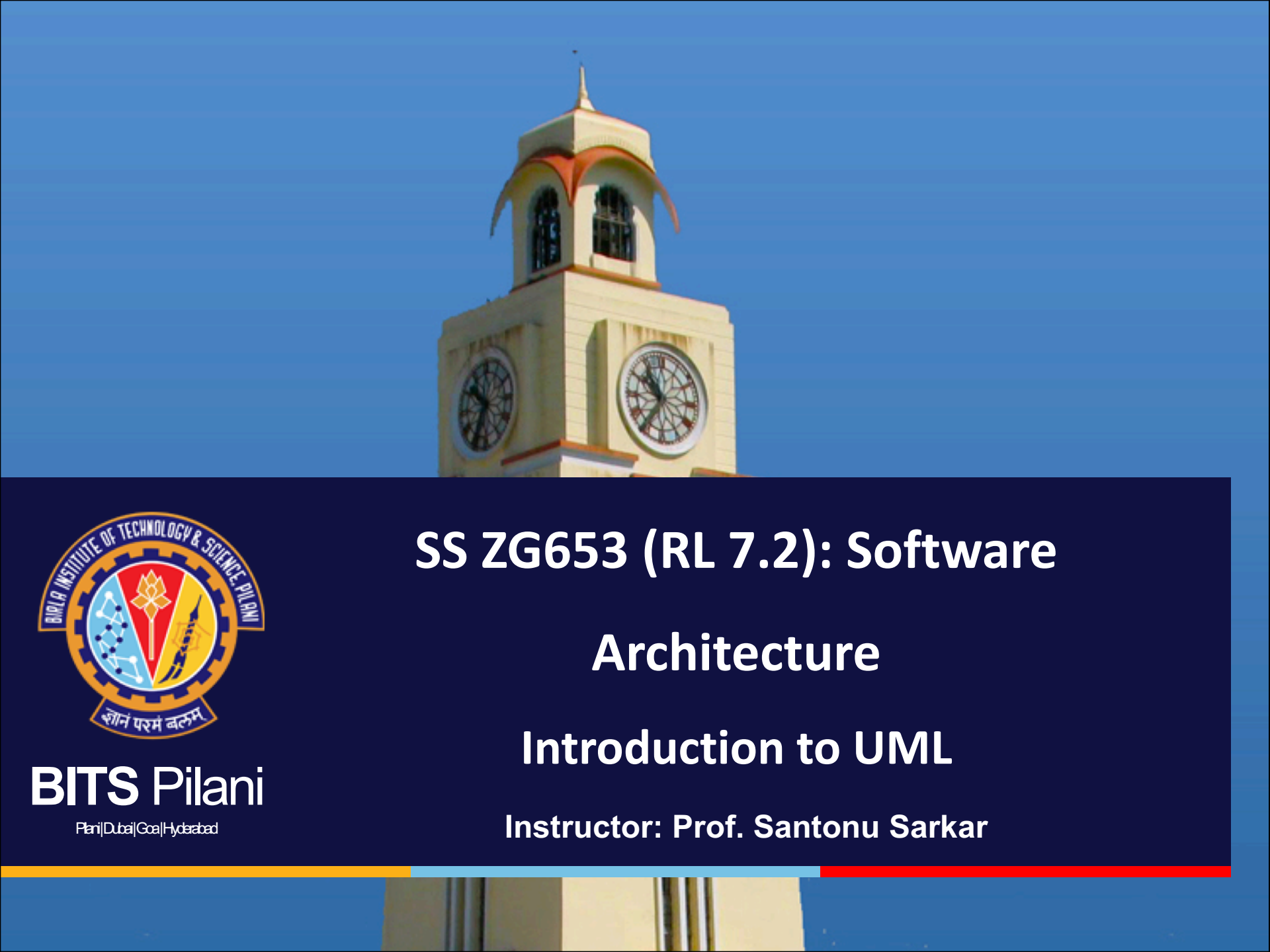
Destroying Objects

- If an object goes “out of scope,” it can no longer be used (its name is no longer known) it is necessary to free the memory occupied by the object
  - Otherwise there will be a “memory leak”

1. Just before freeing the memory
   - It is necessary to perform clean-up tasks (you need to decide) just before getting deleted
     - » Destructor method describes these tasks
     - » Special name for destructor method ~<ClassName> in C++
     - » finalize() method in Java
     - » does not have any return value
2. Then free the memory
   - In C++, we need to explicitly delete this object (delete operator)
   - Java uses references and “garbage collection” automatically.

Thank You

Next class - UML



SS ZG653 (RL 7.2): Software Architecture

Introduction to UML

Instructor: Prof. Santonu Sarkar

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Unified Modeling Language (Introduction)

- **Modeling Language for specifying, Constructing, Visualizing and documenting software system and its components**
- **Model -> Abstract Representation of the system [Simplified Representation of Reality]**
- **UML supports two types of models:**
  - **Static**
  - **Dynamic**

UML

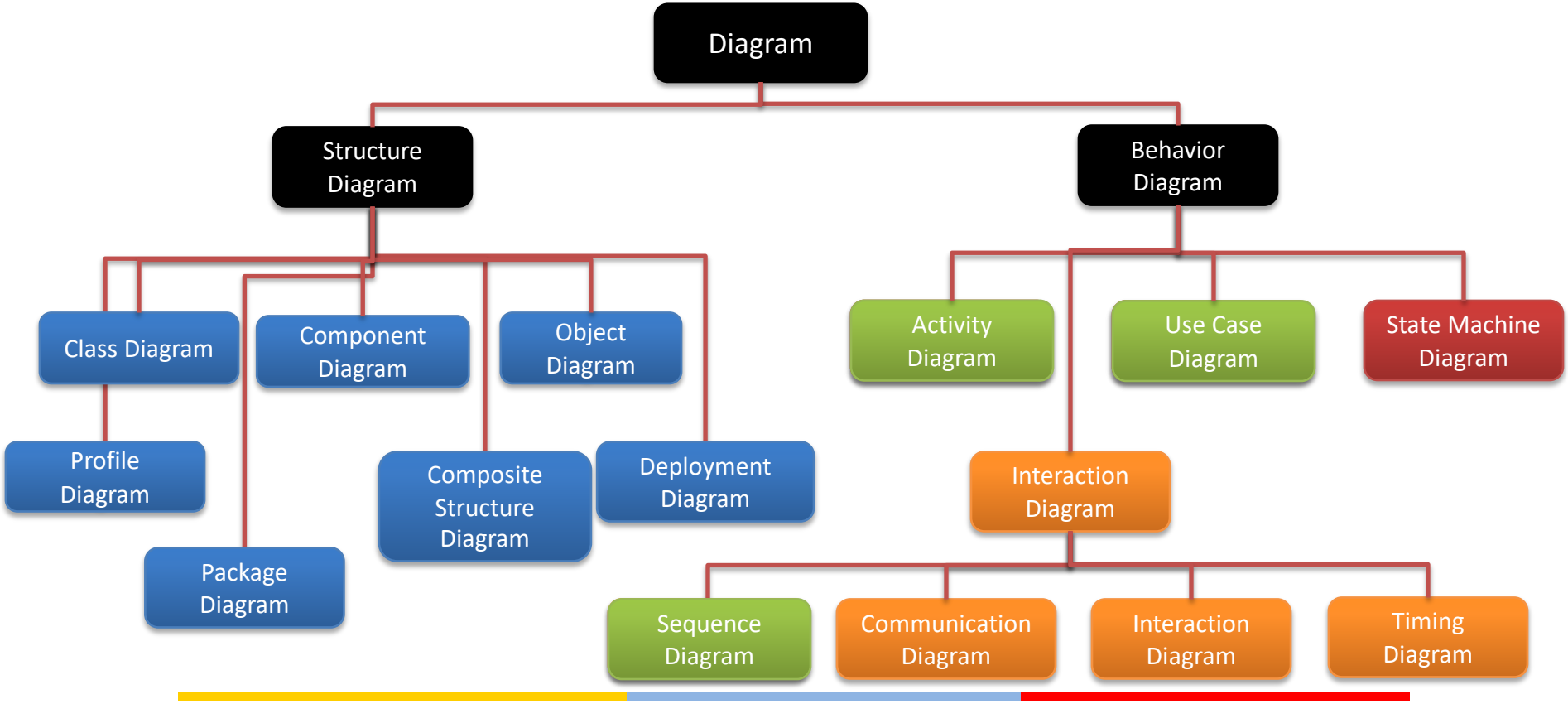
- Unified Modeling Language is a standardized general purpose modeling language in the field of object oriented software engineering

- The standard is managed, and was created by, the Object Management Group.

- UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems

UML Diagrams overview

- Structure diagrams emphasize the things that must be present in the system being modeled- extensively used for documenting software architecture
- Behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems.



Structural Diagrams

- **Class diagram**: system's classes, their attributes, and the relationships
- Component diagram: A system, comprising of components and their dependencies
- Composite structure diagram: decomposition of a class into more finer elements and their interactions
- **Deployment diagram**: describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.
- Object diagram: shows a complete or partial view of the structure of an example modeled system at a specific time.
- Package diagram: describes how a system is split up into logical groupings by showing the dependencies among these groupings.
- Profile diagram: operates at the metamodel level

Behavioral Diagrams

- **Activity diagram**: describes the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.
- **State machine diagram**: describes the states and state transitions of part of the system.
- **Use case diagram**: describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases

Behavioral Model- Interactions

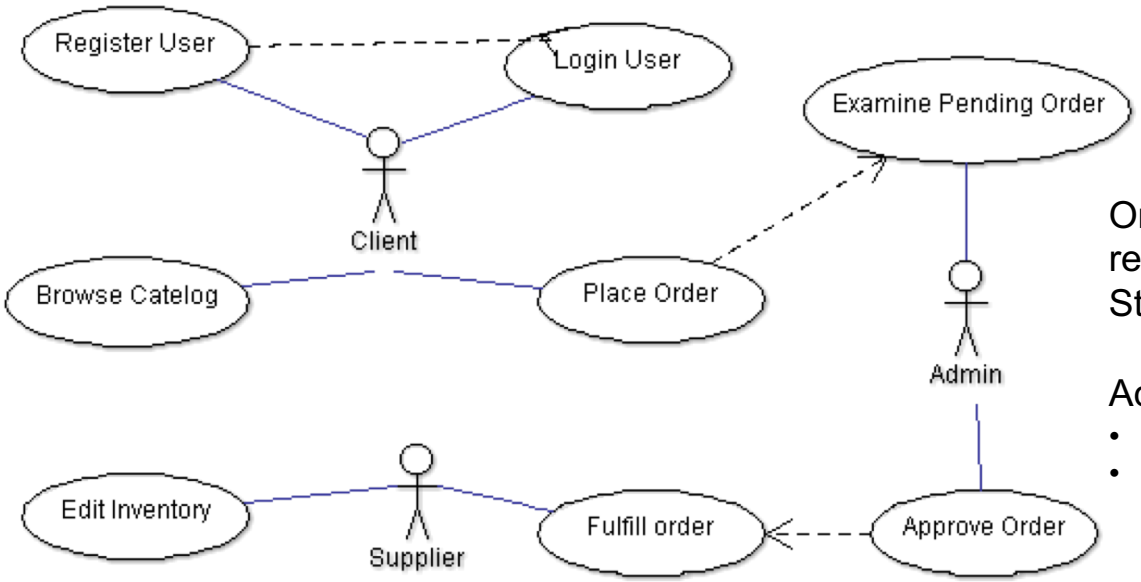
- Communication diagram: shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.
- Interaction overview diagram: provides an overview in which the nodes represent communication diagrams.
- **Sequence diagram**: Interaction among objects through a sequence of messages. Also indicates the lifespans of objects relative to those messages
  - Timing diagrams: a specific type of sequence diagram where the focus is on timing constraints.

Petstore Shopping System

Use Case

Storefront has the main user interface in a Web front-end. Customers use the Storefront to place orders for pets

- Register User
- Login user
- Browse catalog of products
- Place order to OPC (asynchronous messaging)



Order Processing Center (OPC) receives orders from the Storefront.

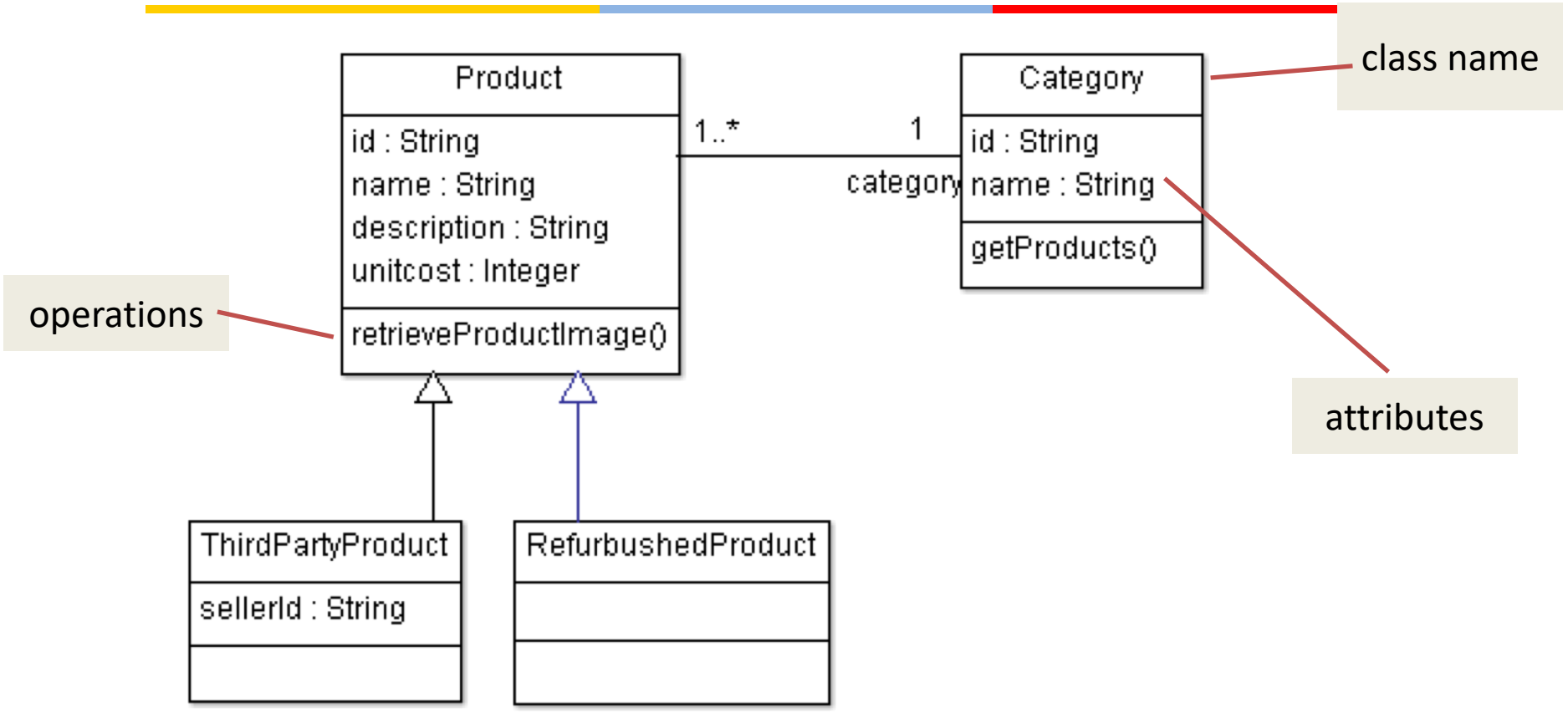
Administrator

- Examine pending orders
- Approve or deny a pending order

Supplier

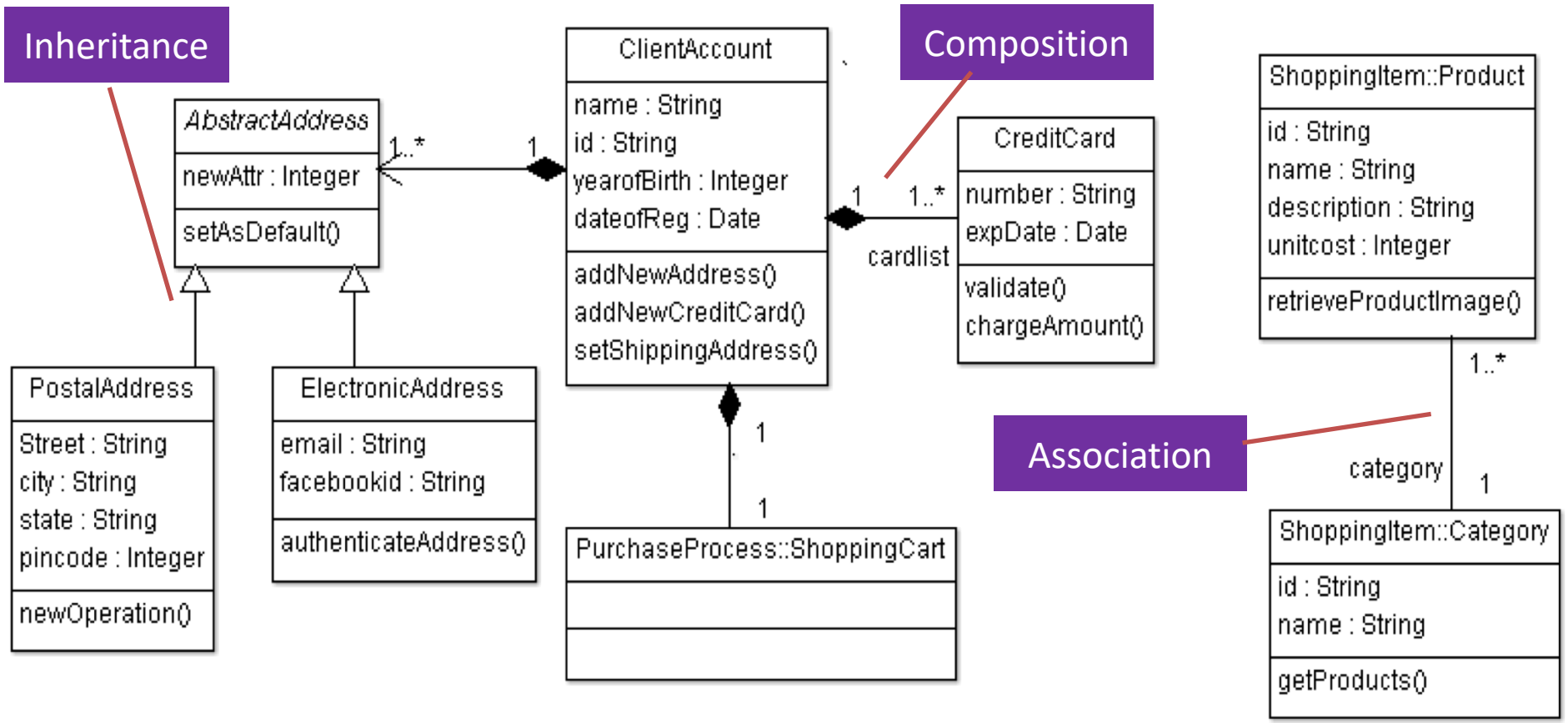
- View and edit the inventory
- Fulfills orders from the OPC from inventory and invoices the OPC.

Class Notation



Visibility Modes : Private(-), Protected (#) , Public (+) and Package Private()

Class Relationships



Generalization Example

- isA, is-a-type of relation ship
- A PostalAddress, or an EmailAddress is an Address
- There can be ThirdPartyProduct or a Refurbished product in the online store

```
public class PostalAddress extends AbstractAddress  {

 private String Street;
 private String city;
 private String state;
 private Integer pincode;
}
```

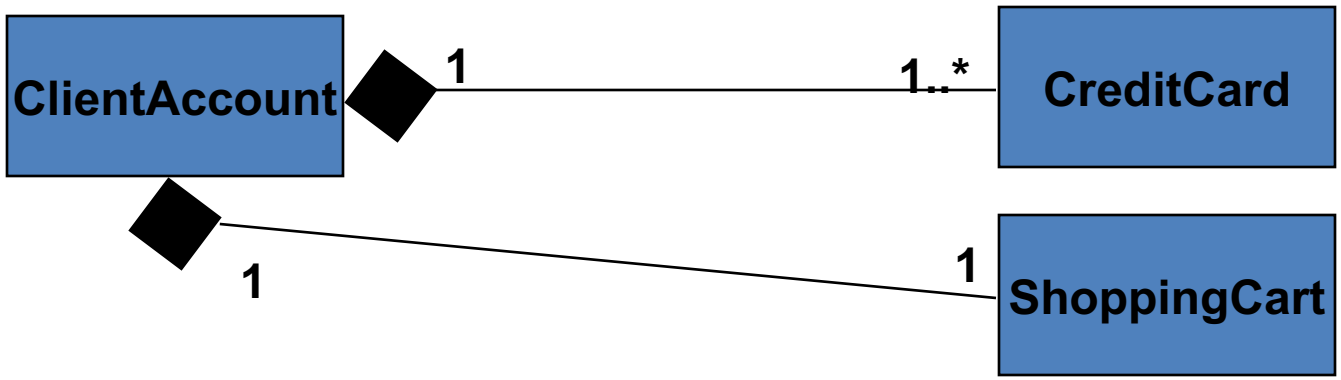
```
public class ElectronicAddress
extends AbstractAddress  {

  private String email;
  private String facebookid;

  public void authenticateAddress() {
  }
}
```

Different forms of association

1. Strongest (Composition)
   - Implies total ownership, if the owner is destroyed, the parts are also destroyed
   - Inner classes will certainly be a composition
2. Aggregation
   - Implies has-a part ownership
   - If the owner is destroyed, the parts still exist
3. Weakest (Association)
   - General form of dependency based on the usage of features

Composition

- Total ownership
- A CreditCard exclusively belongs to one Client, and one client can have many credit cards.
- A client exclusively owns her shopping cart.
- The shopping cart and the credit card will no longer exist if the client is removed



Composition Code snippet

```
public class ClientAccount {
  private String name;
  private String id;
  private Integer yearofBirth;
  private Date dateofReg;
  private Vector  myAbstractAddress;
  private Vector  myCreditCard;
  private ShoppingCart myShoppingCart;

  public void addNewAddress() {  }
  public void addNewCreditCard() {  }
  public void setShippingAddress() {  }
}
```

```
public class CreditCard {
   private String number;
   private Date expDate;
   private ClientAccount
                  myClientAccount;

  public void validate() {   }
  public void chargeAmount() {    }
}
```

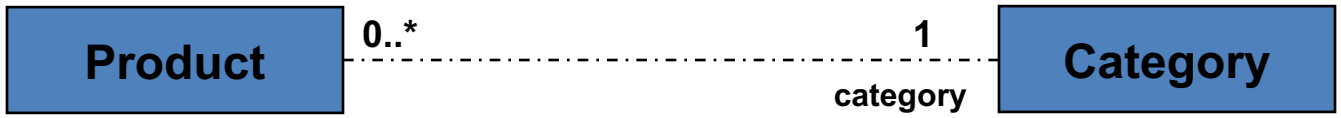
```
public class ShoppingCart {
   private Vector  myShoppingCartController;
   private ClientAccount myClientAccount;
}
```

Aggregation Relationship

- Relatively weaker composition
- Students will exist even when the Professor stops taking the class
- Ducks will exist without the Pond

Association or Dependency Relation

- **A product category in the online shop can have many products**
- **However, a product belongs to only one category.**
- **Both of them independently exist.**

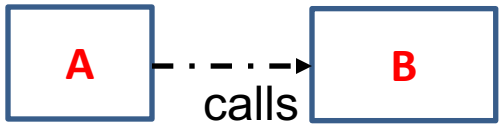



```
public class Product {
   private String id;
   private String name;
   private String description;
   private Integer unitcost;
   private Category category;

  public void retrieveProductImage() {
  }
}
```

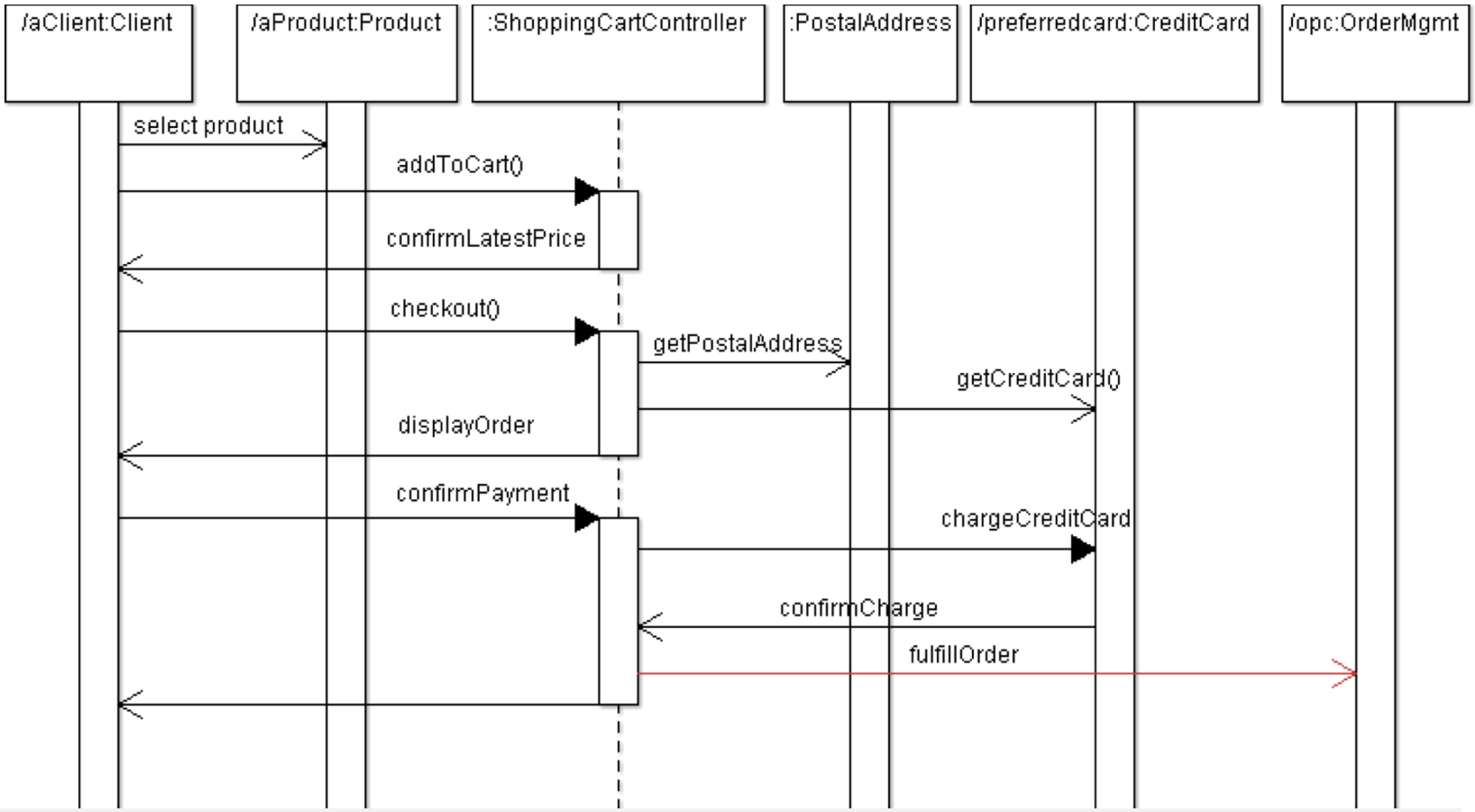
```
public class Category {
   private String id;
   private String name;
   private Vector<Product> myProduct;

   public void getProducts() {
   }
}
```

Dependency Example 2

```
class B
{
  public void doB()
  {
    System.out.println("Hello");
  }
}
class A
{
  public void doS()
  {
    B b1 = new B();
    b1.doB();
  }
} //End of class Test
```



Sequence Diagrams



Sequence diagram is drawn to represent (i) objects participating in an interaction and (ii) what messages have exchanged among those objects

innovate achieve lead

Thank You