Introduction to Object-Oriented Design

Jacek Wasilewski

Object-Oriented Analysis (1)

- **Object–Oriented Analysis** - identifying requirements and developing specifications in terms of object models
- Requirements are organized around **objects**, which integrate **both data** and **functions**
- Modelled after **real-world objects** that the system interacts with

Object-Oriented Analysis (2)

- Tasks of object-oriented analysis:
  - Identifying objects
  - Identifying the hierarchy of objects
  - Identifying objects attributes
  - Identifying object actions
  - Identifying objects interactions

Object-Oriented Analysis (3)

- Three analysis techniques
  - object modelling
  - dynamic modelling
  - functional modelling
- Object modelling – develops the static structure of a system in terms of objects
  - identifies objects, groups into classes, defines relationships between
  - identifies attributes of objects

Object-Oriented Analysis (4)

- Dynamic modelling – describes how individual objects respond to events
  - identifying states
  - identifying events and actions
  - each state expressed in terms of object attributes
  - transitions between states are identified
- Functional modelling – shows how objects interact, how the data changes and is moved between objects/methods.

Object-Oriented Design

- Object–Oriented Design – implementing the results of object-oriented analysis
- Object-oriented analysis creates deliverables not dependent on technology
- Models are converted into solutions using technology

Object-Oriented Programming

- Object-Oriented Programming – paradigm based on objects that store data and interact
- Features:
    - bottom–up design
    - programs organized around objects, grouped in classes
    - focus on data with methods to operate on object’s data
    - interaction between objects through methods
    - reusability of design through creation of new classes by adding features to existing classes

Objects and Classes

- Object is a **real-world** element in an **object–oriented** environment
  - **physical existence** like a customer or a car
  - **conceptual existence** like a project, a process
- A class represents a collection of objects having
  - same characteristic properties
  - common behavior
- Description of the objects that can be created from it
- Object is an instance of a class

Encapsulation and Data Hiding

- Encapsulation - binding both attributes and methods together within a class
- Internal details of a class can be hidden from outside (and should)
- Elements of the class can be accessed from outside only through the interface provided by the class

Polymorphism

- Polymorphism – (Greek) ability to take multiple forms; ability to process objects differently, depending on their data type or class
- Polymorphism implies using operations in different ways, depending upon the instance they are operating upon
- Objects with different internal structures to have a common external interface
- Can be achieved using
  - subtyping
  - composition
  - generics

Inheritance

- Inheritance – new classes to be created out of existing classes by extending and refining capabilities
- Subclass inherits attributes and methods of the super-class, provided that the super-class allows so
- New attributes and methods can be added in the subclass
  - and super-class methods be modified
- Inheritance defines an “is – a” relationship.

Types of Inheritance

- Single inheritance (Java, Scala)
- Multiple inheritance (C++, Scala through traits)
- Multilevel inheritance (class A inherits from B that inherits from C)
- Hierarchical inheritance (classes A, B, C inherit from D)
- Hybrid inheritance (multilevel + hierarchical)

Generalization and Specialization (1)

- Generalization and specialization represent a hierarchy of relationships between classes, where subclasses inherit from super-classes
- In the generalization, the common characteristics of classes are combined to form a class in a higher level of hierarchy, subclasses are combined to form a generalized super-class
- It represents an "is – a – kind – of" relationship
- For example, “car is a kind of land vehicle”, or "ship is a kind of water vehicle"

Generalization and Specialization (2)

- Specialization is the reverse process of generalization
- Distinguishing features of groups of objects are used to form specialized classes from existing classes
- Subclasses are specialized versions of the super-class
- Generalization and specialization typically happens when new classes are added to model new things

Links and Association (1)

- Link - is a representation of a connection between objects
- A link depicts the relationship between two or more objects
- Association – is a group of links having common structure and common behavior
- Association depicts the relationship between objects of one or more classes
- A link can be defined as an instance of an association

Links and Association (2)

- Degree of an association – number of classes involved in a connection
  - unary relationship connects objects of the same class
  - binary relationship connects objects of two classes
  - ternary relationship connects objects of three or more classes
- Cardinality of a binary association - the number of instances participating in an association
  - one–to–one
  - one–to–many
  - many–to–many

Aggregation and Composition

- **Aggregation** or **composition** is a relationship among classes by which a class can be **made up** of any combination of objects of other classes
- Objects can be placed directly within the body of other classes
- **Aggregation/composition** is referred as a “part–of” or “has–a” relationship, with the ability to navigate from the whole to its parts
- Aggregation – child can exist independently
- Composition – child cannot exist independently of the parent

Benefits of Object Model

- The benefits of using the object model
  - faster development of software
  - easier maintenance
  - relatively hassle-free upgrades
  - reuse of objects, designs, and functions
  - reduced development risks, particularly in integration of complex systems

Composition over Inheritance

- Classes should achieve polymorphic behavior and code reuse by **composition instead** of through **inheritance**
- Leads to easier accommodation of future requirements changes
  - that would otherwise require a complete restructuring of business-domain classes in the inheritance model
- Avoids problems often associated with relatively minor changes to an inheritance-based model that includes several generations of classes
- Does not mean we only use composition
  - more like – "do not use inheritance for everything"

References

- http://www.tutorialspoint.com/object_oriented_analysis_design/index.htm
- https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
- https://en.wikipedia.org/wiki/Single_responsibility_principle
- https://en.wikipedia.org/wiki/Composition_over_inheritance