03-10-2023 Object Oriented Concepts

03 October 2023 10:18

Need of object oriented Approach
State of object
Behavior of an object
Identity of an object
Responsibility of object
Examples of Object

Object Model: Abstraction
Abstraction of a person object

Object Model: Encapsulation

Object Model: Modularity (breaking in parts)

Object Model: Inheritance (handling complexity, Property of a class to access super class attribute)

Object Model: Containment

Object Model: Polymorphism (Same name different attribute)

Object Model: Minor Pillars (3 : Strong Type Casting, #Persistence

History of java
Feature of java [Simple, Distributed, Robust, Secure, Architecture Neutral

C program Execution
Java program Execution

A Typical java Environment

Class Loader: [Loads required .class file in memory.]

Bytecode Verifier

Java Virtual Machine (JVM) [Platform- depended]

Interpreted [Java is Interpreter & java run time system]

Multithreaded

Portable

Class [Object is the Instance of Class]

Exmple :

class name with Pascal case{

Member variables are private and camel case

default constructor as public
parameterized contractor as public

setters as public and camel case
getters as public and camel case

toString as public and camel case
Behaviors as public and camel case

}

28-Sep-2023 Git & Github

03 October 2023 10:19

28/Sep/2023 ----------

Github
- Github is a "Hub" (a place or platform) where Git users build software together.
- GitHub is also an hosting provider and version control platform you can use to collaborate on open source projects and share files.
- When you're using GitHub, you're working with Git beneath the hood.
- developed in 2008
- Managed by Microsoft
- Open Source
- Centralized

Git
- developed in 2005 by Linus Torvals
- Open Source for tracking chnages in a distributed version control system.
- No central server(Decentralized)
- Git is a distributed version-control system (DVCS) that allows multiple developers or other contributors to work on a project.
- It provides a way to work with one or more local branches and push them to a remote repository.
- Git is responsible for everything GitHub-related that happens locally on your computer.

Key Git features include:
- It's installed and used on your local machine Handles ,
- version control Supports branching

To learn more about Git, see Using Git.
- GitHub is a cloud platform that uses Git as its core technology.
- It simplifies the process of collaborating on projects and provides a website, command-line tools, and overall flow that allows developers and users to work together.
- GitHub acts as the "remote repository" mentioned previously in the Git section.

Git commands:

1. **git init**:
   - Initializes a new Git repository in the current directory.

2. **git clone [repository URL]**:
   - Creates a copy of a remote Git repository on your local machine.

3. **git add [file]**:
   - Stages changes for commit. You can use `git add .` to stage all changes in the current directory.

4. **git commit -m "[commit message]"**:
   - Commits staged changes with a descriptive message.

5. **git status**:
   - Shows the status of your working directory, including untracked, modified, and staged files.

6. **git log**:
   - Displays the commit history of the current branch.

7. **git branch**:
   - Lists all local branches, with the current branch highlighted.

8. **git checkout [branch name or commit hash]**:
   - Switches to a different branch or commit.

9. **git merge [branch name]**:
   - Merges changes from one branch into the current branch.

10. **git pull**:
    - Fetches changes from a remote repository and merges them into the current branch.

11. **git push**:
    - Pushes your local commits to a remote repository.

12. **git remote -v**:
    - Lists all remote repositories connected to your local repository.

13. **git fetch [remote]**:
    - Fetches changes from a specific remote repository.

14. **git reset [commit]**:
    - Unstages changes from the commit and keeps them in the working directory.

15. **git diff**:
   - Shows the differences between your working directory and the last commit.

16. **git stash**:
   - Temporarily saves changes that are not ready to be committed, allowing you to switch branches.

17. **git tag [tag name]**:
   - Creates a tag for a specific commit to mark important points in history.

18. **git remote add [remote name] [repository URL]**:
   - Adds a remote repository with a specified name and URL.

19. **git checkout -b [new branch name]**:
   - Creates a new branch and switches to it.

20. **git rm [file]**:
   - Removes a file from both the working directory and the Git repository.

21. **git mv [old file] [new file]**:
   - Renames or moves a file in your Git repository.

03-10-2023 UML (Unified Modeling Language)

03 October 2023 10:33

UML (Unified Modeling Language)

-> OMG standard, Object Management Group

-> Based on work from Booch, Rambaugh, Jacobson

UML is a modeling language to express and design documents, software

-> Particularly useful for OO design

-> Not a process, but some have been proposed using UML

-> Independent of implementation language

Why use UML?

-> Open Standard, Graphical notation for

-> Specifying, visualizing, constructing, and documenting software systems

-> Language can be used from general initial design to very specific detailed design across the entire software development lifecycle

-> Increase understanding/communication of product to customers and developers

-> Support for diverse application areas

-> Support for UML in many software packages today (e.g. Rational, plugins for popular IDE's like NetBeans, Eclipse)

-> Based upon experience and needs of the user community

Contributions to UML

-> Inundated with methodologies in early 90's Booch, Jacobson, Yourden, Rumbaugh. Booch, Jacobson merged methods 1994. Rumbaugh joined 1995

1997 UML 1.1 from OMG includes input from others, e.g. Yourden

UML v2.0 current version

Systems, Models and Views

-> A model is an abstraction describing a subset of a system

-> A view depcits selected aspects of a model.

-> A notation is a set of graphical or textual rules for depicting views

-> Views and models of a single system may overlap each other.

UML Baseline

Static

-> Use Case Diagram

-> Class Diagram

Dynamic

-> Package Diagram

-> Object Diagram

-> State Diagram

-> Activity Diagram

-> Sequence Diagram

-> Collaboration Diagram

Implementation

-> Component Diagram

-> Deployment Diagram

1. Use Case Diagram

-> A use case represents a class of functionality provided by the system as an event flow.

A use case consists of:

-> Unique name

-> Participating actors

-> Entry conditions

-> Flow of events

-> Exit conditions

-> Special requirements

2. Class Diagram

->

Relationships

-> Association ---------- solid line

-> Aggregation ----------<> solid line

-> Composition ----------<*> solied line with filled arrow

-> Generalization <|---------- solid line

-> Realization <|---------- Actual dotted

Lines or arrows between classes indicate relationships

Association

-> A relationship between instances of two classes, where one class must know about the other to do its work,

e.g. client communicates to server indicated by a straight line or arrow

Aggregation

-> An association where one class belongs to a collection, e.g. instructor part of Faculty

Indicated by an empty diamond on the side of the collection

Composition

-> Strong form of Aggregation

Lifetime control; components cannot exist without the aggregate

Indicated by a solid diamond on the side of the collection

Inheritance

-> An inheritance link indicating one class a superclass relationship, e.g. bird is part of mammal

Indicated by triangle pointing to superclass

Binary Association

A knows about B , But B nows nothing about A.

Inheritance

27 - Sep - 2023 ----------

Object Diagram:
bject is the entity is veiled define structure and behavior

Characteristics:
1)State, 2)Behaviour, 3)identity ,4)Responsibility

Define the Object:
State: current value of attribute.
Behaviour: All Methods
identity: Uniqueness
Responsibility: what kind of role which play its called responsibility.

4 Piller:
Abstraction
Incapsulation
Inheritance
Polymorphisum

3 Mejor Piller:
Type Casting

Package Diagram:
Collection of logically related UML elements

Interaction Diagram
Sequence Diagram

03-10-2023 UML

03 October 2023 10:34

(UML)Unified Modeling Language
OMG(Object Management Group)
Based on work from Booch, Rumbaugh, Jacobson
UML is a modeling language to express and design documents, software
Particularly useful for OO design
Not a process, but some have been proposed using UML
Independent of implementation language

Lines or arrows between classes indicate relationships
Association
A relationship between instances of two classes, where one class must know about the other to do its work, e.g. client communicates to server
indicated by a straight line or arrow
Aggregation
An association where one class belongs to a collection, e.g. instructor part of Faculty
Indicated by an empty diamond on the side of the collection
Composition
Strong form of Aggregation
Lifetime control; components cannot exist without the aggregate
Indicated by a solid diamond on the side of the collection
Inheritance
An inheritance link indicating one class a superclass relationship, e.g. bird is part of mammal
Indicated by triangle pointing to superclass

Open Standard, Graphical notation for
Specifying, visualizing, constructing, and documenting software systems
Language can be used from general initial design to very specific detailed design across the entire software development lifecycle
Increase understanding/communication of product to customers and developers
Support for diverse application areas
Support for UML in many software packages today (e.g. Rational, plugins for popular IDE's like NetBeans, Eclipse)
Based upon experience and needs of the user community

Inundated with methodologies in early 90's
Booch, Jacobson, Yourden, Rumbaugh
Booch, Jacobson merged methods 1994
Rumbaugh joined 1995
1997 UML 1.1 from OMG includes input from others, e.g. Yourden
UML v2.0 current version

Use Case Diagrams
Class Diagrams
Package Diagrams
Interaction Diagrams
Sequence
Collaboration
Activity Diagrams
State Transition Diagrams
Deployment Diagrams

A model is an abstraction describing a subset of a system
A view depicts selected aspects of a model
A notation is a set of graphical or textual rules for depicting views
Views and models of a single system may overlap each other

Examples:
System: Aircraft
Models: Flight simulator, scale model
Views: All blueprints, electrical wiring, fuel system

'

A <<includes>> B
Somewhere through use case A, use case B is called. When B finishes, A carries on from where it left off.
A <<extends>> B
All the steps from use case A are performed during the execution of use case B, at the extension point which is specified within B.
A <<precedes>> B
Use case A must take place in its entirety before use case B even begins.
A <<invokes>> B
Use case B happens during the lifespan of use case A.

Name: Purchase ticket

Participating actor: Passenger

Entry condition:
Passenger standing in front of ticket distributor.
Passenger has sufficient money to purchase ticket.

Exit condition:
Passenger has ticket.

Event flow:
1. Passenger selects the number of zones to be traveled.
2. Distributor displays the amount due.
3. Passenger inserts money, of at least the amount due.
4. Distributor returns change.
5. Distributor issues ticket.

A use case represents a class of functionality provided by the system as an event flow.

A use case consists of:
Unique name
Participating actors
Entry conditions
Flow of events

Exit conditions
Special requirements

An actor models an external entity which communicates with the system:
User
External system
Physical environment
An actor has a unique name and an optional description.
Examples:
Passenger: A person in the train
GPS satellite: Provides the system with GPS coordinates

Used during requirements elicitation to represent external behavior

Actors represent roles, that is, a type of user of the system
Use cases represent a sequence of interaction for a type of functionality; summary of scenarios
The use case model is the set of all use cases. It is a complete description of the functionality of the system and its environment

https://www.visual-paradigm.com/features/uml-tool/

STATIC
Use case diagram
Class Diagram
Dynamic
Object Diagram
State Diagram
Activity Diagram
Sequence Diagram
collaboration Diagram

Implementation

Component Diagram
Deployment Diagram

04-10-2023 Java Practical Constructor Section

04 October 2023 10:54

1. **Default Constructor**:
   - A default constructor is a constructor that is automatically provided by Java if you do not define any constructors in your class.
   - It has no parameters and does not accept any arguments during object creation.
   - The default constructor initializes instance variables to their default values (e.g., 0 for numeric types, null for reference types).
   - It is used when you create an object of the class without specifying any constructor arguments.

2. **Parameterized Constructor**:
   - A parameterized constructor is a constructor that accepts one or more parameters as arguments during object creation.
   - It allows you to initialize instance variables with specific values at the time of object creation.
   - You can define multiple parameterized constructors in a class with different parameter lists to provide flexibility in object initialization.

Use getter and setter methods in this java program =>

Fields name-
Emp_Id;
Emp_name;
Emp_salary;

Add 1st switch case to add record

Add 2nd switch case to display record

Add 3rd switch case to delete record

Add 4th switch case is Replace/Update record
    Add nested switch case for name and salary changes

02-10-2023 Agile methodology

04 October 2023 15:15

Agile methodology is a set of principles and practices for software development and project management that prioritize flexibility, collaboration, and customer satisfaction. Here are some basic things to know about Agile:

1. **Iterative and Incremental:** Agile projects are divided into small increments or iterations, with each iteration delivering a potentially shippable product increment. This allows for continuous improvement and adaptation.

2. **Customer-Centric:** Agile places a strong emphasis on customer feedback and involvement throughout the development process. Customer needs and priorities are considered and integrated into the product.

3. **Collaborative:** Agile teams are typically cross-functional, self-organizing groups that work closely together. Collaboration among team members, stakeholders, and customers is encouraged.

4. **Embracing Change:** Agile methodologies are adaptive and open to change. Changes in requirements, priorities, or technology can be accommodated even late in the project.

5. **Value Delivery:** The goal of Agile is to deliver value to the customer quickly and frequently. Features or functionality that provide the most value are prioritized and delivered first.

6. **Continuous Testing and Integration:** Agile teams practice continuous testing and integration to ensure that the software remains functional and reliable throughout development.

7. **Transparency:** Agile promotes transparency in project progress, with regular status updates and open communication within the team and with stakeholders.

8. **Scrum, Kanban, and Lean:** There are different frameworks and methodologies within the Agile umbrella, including Scrum, Kanban, and Lean. Each has its own practices and principles but shares the core values of Agile.

9. **Scrum Roles:** In Scrum, one of the most popular Agile frameworks, there are three main roles: Scrum Master (responsible for facilitating the Scrum process), Product Owner (represents the customer and sets priorities), and Development Team (responsible for building the product).

10. **Sprints:** In Scrum, work is organized into time-boxed iterations called "sprints," usually lasting 2-4 weeks. At the end of each sprint, a potentially shippable product increment is delivered.

11. **Backlog:** Agile projects maintain a backlog, which is a prioritized list of features, user stories, or tasks. The backlog is continually refined and reprioritized based on changing requirements and feedback.

12. **Retrospectives:** Agile teams regularly hold retrospectives to reflect on their process and identify areas for improvement. This feedback loop helps teams continuously enhance their performance.

13. **Velocity:** Teams often use velocity as a metric to measure how much work they can complete in a sprint. It helps with predictability and planning.

14. **Distributed and Remote Teams:** Agile practices can be adapted for distributed or remote teams, with the use of collaboration tools and effective communication.

15. **Agile Manifesto:** The Agile Manifesto, written by a group of software developers in 2001, outlines the core values and principles of Agile. It includes values such as "Individuals and interactions over processes and tools" and "Responding to change over following a plan."

Agile is a flexible and adaptable approach to project management and product development, and its principles can be applied to various industries and types of projects, not just software development. However, it's important to note that implementing Agile effectively often requires training, practice, and a cultural shift within an organization to fully embrace its principles and benefits.

02-10-2023 Jira

04 October 2023 15:22

- JIRA is a tool developed by Australian Company Atlassian. It is used for bug tracking, issue tracking, and project management. The name "JIRA" is actually inherited from the Japanese word "Gojira" which means "Godzilla".
- The basic use of this tool is to
  - track issue and bugs related to your software and Mobile apps.
  - It is also used for project management.
- The JIRA dashboard consists of many useful functions and features which make handling of issues easy.