Events

* Design an event.
* take out the nouns of the problem.
* and make them into classes.
* take verbs out the action to events.

Problem 1

Bank

* Customers
* Account
* Message
  + Email
  + SMS
* Actions
  + Withdraw
  + Deposit
  + Send SMS
  + Send Email

Types of Classes

* Control
  + Controls flow of application.
  + Types
    - Request Handler Controller
      * Handles request from client.
    - Business Controller
      * Business Logic is written in this.
    - Repositories Controller
      * Takes care of connection to database.
      * Communicates with the databases.
    - User Defined Controllers
    - Log Managers Controllers
    - Error Controller
      * Handle the error and exception.
    - Audit Controller
      * Audit the log of Application.
    - Model View Controllers (Very Important and useful)
* Entity
  + Contains data alone.
  + Entities are durable and persistent.
  + RDBMS or files are entities.
  + Save in json or table, transfer different layers of the software from object by json or xml and covert to object. Now the object of entity mapped to table.
  + No of entities that can be created is equivalent to the no of rows in the table.
  + Entity classes are serializable.
* Boundary
  + A class that interacts with external environment is called boundary classes.
  + They are views or UI’s.
  + Ex. From a user interface like giving inputs in console, forms, etc.
    - Client-side need Adapter.
    - Server-side need Proxy.

Types of Objects

* Actors
  + Extract service from other objects they don’t provide service.
* Agents
  + Take service from other objects and provide them.
  + delegations possible between objects
* Servers
  + Provide a service for other objects.
  + It is self-contained.

Event-driven programming is a design approach where the interaction between components or objects is based on the occurrence of events rather than direct method calls. This promotes loose coupling and late binding, making applications more maintainable and scalable. The concept is explained through an analogy involving traffic signals and police constables:

**Key Concepts:**

1. **Event-Driven Programming:**
   * Interaction between components is based on events.
   * Components communicate through events rather than direct method calls.
   * Increases modularity and reusability of components.
2. **Loose Coupling:**
   * Components are designed to have minimal dependencies on each other.
   * Changes to one component don't heavily impact others.
   * Promotes flexibility and easier maintenance.
3. **Late Binding:**
   * Binding of functions (methods) occurs at runtime, not compile time.
   * Components are brought together through events dynamically.
4. **Publisher-Subscriber Model:**
   * In the analogy, traffic signals (publisher) and drivers (subscribers) are used.
   * Publishers raise events, and subscribers react to those events.
   * Communication is indirect; publishers and subscribers are unaware of each other.

**Traffic Signal Analogy:**

1. **Traffic Signal: Publisher**
   * Represents an event, such as changing from green to red.
   * The state of the road (empty/full) determines the signal change.
   * "Publisher" in this analogy.
2. **Police Constable: Event Raiser**
   * Raises events (changes the traffic signal) based on the state of the road.
   * Acts as an intermediary between the publisher (signal) and subscribers (drivers).
   * Controls traffic without direct interaction with drivers.
3. **Drivers: Subscribers**
   * React to signal changes without knowing the signal's source.
   * Don't directly communicate with the police constable or other drivers.
   * "Subscribers" in this analogy.

**Implementing in Code:**

1. **Delegates and Events:**
   * Delegates represent methods.
   * Events are based on delegates and represent occurrences (signals).
2. **Loose Coupling in Events:**
   * Account controller and message controller are independent.
   * Account controller raises events when actions (deposit/withdraw) occur.
   * Message controller subscribes to these events and reacts accordingly.
   * These components remain independent yet collaborate through events.
3. **Late Binding with Events:**
   * Events (delegates) are assigned and raised at runtime using **+=**.
   * Allows components to work together without being aware of each other.

**Benefits:**

1. **Modularity:** Components are isolated, making them easier to develop, test, and maintain.
2. **Scalability:** New components can easily interact with existing ones through events.
3. **Flexibility:** Changes to one component's implementation won't disrupt others.

**Conclusion:**

Event-driven programming promotes loose coupling and late binding by allowing components to interact through events. This design approach enhances maintainability and scalability, as well as enabling components to remain independent while collaborating effectively. It also encourages modularity and flexibility, making it an advantageous programming paradigm.

Working with Threads

What is a thread? (In Friday test)

* A Thread is an independent path of executable code waiting for CPU time.
* **Threads** are the basic unit to which an **operating** **system allocates processor time**.
* More than one thread can be executing code inside a **process**.
* Operating systems use **processes** to logically separate different applications that they are executing.
* Each thread maintains,
  + Exception handlers
  + A Scheduling priority
  + A set of structures the system uses to save the thread context until it is scheduled.
* The scope of thread is limited to the process in which it is created. (See pic)

/\*

A thread is an independent path of executable code writing for CPU time

    THreads are the basic unit to which an operating system allocates the process time

    More than one thread can exist within the same process

    OS use  processes to logically separate the resources used by different running programs

    that thread that contain

    1.Exception Handler

    2.a Scedhuliing priority,and

    3.a set of structures the kernel uses to save the thread context until it is scheduled

    resource alloccate in a process can access the other process.

    single core processor can only run one thread at a time

    scope of thread is limited to the process created

    thread can be created by using two mechanism

    1.using thread class

    2.using thread pool

    jvm load the three thread minimum

    1.main thread

    2.garbage collector thread

    3.jit compiler thread

\*/

Thread Priority

* Thread Priority is a Enum.
* Every thread has a priority.
* Threads with higher priority are executed in preference to threads with lower priority.
* When code running in some thread creates a new Thread object, the new thread has its priority initially set equal to the priority of the creating thread.

Multithreading

* When multiple threads (2 or more threads) execute byte-code instruction sequences in the same program at the same time, then that action is known as multithreading.
* Multithreaded applications are faster and efficient when they are running on a multiprocessor machine, or multi core machine than on a single core or single processor.

Threads synchronized

Can be asynchronized also

So we use tasks