20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by Sudeep V
CB.EN.U4CYS21073
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

AMRITA
VISHWA VIDYAPEETHAM

Outline

Declarative paradigm:

Definition: - Abstraction of control flow and logic for software execution.Emphasizes stating tasks or desired outcomes over step-by-step instructions.Commonly used in databases, configuration management, and DSLs.

Execution in Declarative Programming: - Relies on language components to execute steps toward a specified outcome.

Integration with Imperative Programming: - Some languages, like Java, support a combination of declarative and imperative approaches.Annotations in Java allow the addition of declarative capabilities to imperative code.

Mechanisms in Declarative Programming - Involves constraints and logic to define setup and desired outcomes.Uses DSLs to separate control flow from logic within the language.

Declarative paradigm

Practical Applications: Commonly used in relational databases with SQL (e.g., SELECT * FROM <TableName>).Configuration management tools (e.g., Chef, Puppet) employ DSLs for defining tasks.

Contrast: Imperative vs. Declarative Code Example:

- Imperative code explicitly outlines steps, checks conditions, and specifies actions.

- Declarative code, using Chef DSL, states desired system state without explicit step details.

Comparison between Declarative Programming and Imperative Programming

Listing 1: Imperative Programming Code

```
if file_does_not_exist_on_server:
    copy_file_to_server()
else:
    if file_is_older_on_server:
        overwrite_file_on_server()
```

Listing 2: Declarative Programming Code

```
file '/path/to/file.txt' do
  source 'file.txt'
  action :create
end
```

Declarative paradigm

Summary

-Declarative programming abstracts control flow, emphasizing the desired outcome over explicit step-by-step instructions.

-It integrates with imperative programming, and practical applications include databases and configuration management tools.

-Using DSLs, declarative code, as exemplified by Chef, enhances code simplicity by focusing on the desired state rather than explicit procedural details.

SQL as a Declarative Language

-SQL is a declarative programming language closely tied to the relational database model. It serves as an accessible and widely adopted tool for query, data modification, and data definition, including the definition of data structures such as tables.

-In the realm of data management, SQL plays a crucial role, embraced by developers, data analysts, and businesses alike. In contrast to procedural languages like C or Java, SQL follows a declarative approach. Rather than explicitly outlining steps, users express intent without specifying exact procedures. This emphasis on desired outcomes simplifies complex queries and data manipulation.

-The declarative nature of SQL brings notable advantages. Firstly, it streamlines the coding process by focusing on desired outcomes rather than intricate methods. Secondly, it abstracts complexities, making it easier for developers to work with databases without delving into algorithmic details. Lastly, declarative languages like SQL often exhibit greater scalability, effortlessly adapting to changes in database structure or data volume without extensive code modifications.

Paradigm 1 - SQL

SQL's Role in Relational Databases:

Querying: Retrieve, filter, sort, and perform complex operations with concise SQL statements.

Data Modification: Insert, update, and delete data with declarative specifications.

Data Integrity: Define constraints, relationships, and data types for maintaining data accuracy.

Security: Manage access permissions and privacy using DCL commands.

Transactions: Ensure consistency and integrity with TCL commands.

Paradigm 1 - SQL

SQL Components:

Data Definition Language (DDL): CREATE TABLE, ALTER TABLE, DROP TABLE.

Data Manipulation Language (DML): SELECT, INSERT, UPDATE, DELETE.

Data Control Language (DCL): GRANT, REVOKE.

Transaction Control Language (TCL): COMMIT, ROLLBACK, SAVEPOINT.

Paradigm 1 - SQL

Tables and Select:

-CREATE TABLE statement for table creation.
-SELECT statement for basic and advanced querying. -Examples of table creation, insertion, and selection using SQL statements.

JOINS in SQL:

-Essential for extracting data from multiple tables.
-Types include CROSS JOIN, SELF JOIN, INNER JOIN, OUTER JOIN (LEFT, RIGHT, FULL OUTER JOIN).
-Illustrative examples for each join type.

Interpreting SQL:

-This section guides users in understanding system-generated InterSystems SQL Query Plans. When compiling a SQL query, instructions are generated based on the SQL compiler's understanding of table structures, optimizing for efficiency.

-The query access plan provides a human-readable translation, helping authors visualize data access. Authors can modify queries based on insights, especially regarding sub-queries, JOINs, and UNIONs.

-The plan may indicate NULL value creation in OUTER JOINs and result row amalgamation in UNIONs. To generate a query execution plan, the EXPLAIN command is used.

Paradigm 1 - SQL

Advantages of Declarative Programming with SQL:

- Simplifies coding by emphasizing desired outcomes.

- Abstracts complexities, easing work with databases.

- SQL engines feature query optimizers for efficient fine-tuning.

- Exhibits scalability, adapting to structural or volume changes without extensive code modifications.

Paradigm 1 - SQL

Real-World Applications:

Web Development: Fundamental for user authentication, content management, and e-commerce.

Data Analysis: Used by analysts for insights from large datasets.

Business Intelligence: Central to creating reports, dashboards, and data visualizations.

Database Administration: Critical for managing structures, optimizing performance, and ensuring integrity.

Mobile Apps: Utilized for local data storage, facilitating offline functionality and server synchronization.

Paradigm 1 - SQL

Conclusion:

- SQL's declarative nature streamlines database interactions.

- Prioritizes outcomes over procedural details, crucial for data professionals.

- Declarative programming abstracts control flow, fostering problem-focused development.

- SQL's real-world applications extend to various domains, making it an essential tool for efficient data-driven decision-making.

Event driven

Events:

Triggers or Notifications: Serve as indicators of significant occurrences during program runtime.

Diverse Examples: Include user interactions such as clicks, keystrokes, sensor inputs, and incoming messages.

Event Handler:

Code Association: Represents a dedicated piece of code linked to a specific event.

Action Specification: Encapsulates actions or computations, providing a blueprint for the desired response to the associated event.

Central Role: At the core of event-driven programming, event handlers dictate the program's behavior based on encountered events.

Flexibility: Allows the program to dynamically respond to various scenarios, enhancing adaptability.

Event driven

Event Loop:

Fundamental Component: Orchestrates the functioning of event-driven systems.

Continuous Scanning: Constantly checks for events and dispatches relevant event handlers.

Responsiveness Assurance: Ensures the program remains responsive to its dynamic environment.

Multithreading Support: Adeptly handles multiple events concurrently, suitable for asynchronous environments.

Callbacks:

- Functions executed in response to specific events.

- Enhance modularity and flexibility in program design.

Event driven

Benefits of event driven paradigm

Responsiveness:

- Prompt reaction to unfolding events.

Modularity:

- Separation of event handlers from main logic.

- Promotes code organization and maintainability.

Concurrency:

- Adept handling of multiple events concurrently.

- Suitable for multi-threaded or asynchronous environments.

Flexibility:

- Seamless modification of program behavior with adjustments to event handlers.

Event driven

Challenges:

Complexity:

- Managing intricate event flows, especially in large-scale applications.

Debugging:

- Challenges due to the asynchronous nature of events.
- Requires careful consideration and testing.

Ordering:

- Sequence of events influences program behavior.
- Potential for race conditions or unexpected outcomes.

Event driven

Real-World Examples:

GUI Applications:

- Tkinter (Python), Swing (Java), Windows Forms (C), React (JavaScript).

- Handles user input events for responsive interfaces.

Server-side Applications:

- Node.js (Express), Django/Flask (Python), Ruby on Rails, ASP.NET Core (C).

- Efficiently manages requests, performs tasks, and handles database interactions.

Real-time Systems:

- WebSocket, Socket.io, RabbitMQ, Apache Kafka.

- Applied in online gaming, stock trading, and IoT for real-time data processing.

Paradigm 2 - Javascript

Introduction:

-JavaScript is a versatile programming language that allows you to add interactivity, perform calculations, and create rich web applications, making it an essential tool in web development.

-When a user opens a webpage, the browser interprets the HTML, CSS, and JavaScript code found within the source of the document.

-JavaScript is a high-level scripting language that conforms to the ECMAScript specification and is primarily used for creating dynamic content within web applications.

Paradigm 2 - Javascript

JavaScript is often used to enhance a user's experience on a website by providing:

- Dynamic content updates
- Interactive forms and validation
- Animations and transitions
- Asynchronous data loading without page refresh
- Client-side processing and calculations
- Creating complex web applications

Paradigm 2 - Javascript

JavaScript is not limited to web development. It can also be used in:

- Server-side programming using Node.js
- Middleware and APIs using Express.js
- Mobile application development using React Native, Ionic, or NativeScript
- Desktop application development using Electron

JavaScript events are triggered by user actions or API-generated signals. Event handling involves creating listeners with ‘addEventListener‘ for elements, and removal uses ‘removeEventListener‘. A sample event listener responds to a button click, displaying event details. Various events, such as drag-and-drop, mouseenter, keydown, and scroll, offer diverse interactions.

Events

In programming, events are things that happen or occur while a program is being executed. A variety of activities, including user actions, system notifications, and external signals, might cause these events to occur. Building responsive and engaging software requires an understanding of events.

Categories of Events:

1. Events via the User Interface (UI):

- Click Event: Started when the mouse was clicked.
- Double Click Event: Two rapid mouse clicks in quick succession activate this event.
- Mouseover/Mouseout Events: Activated upon the mouse's entry or departure from a feature.
- Keydown/Keyup Events: Records the pressing and letting go of keys.

2. Configuration Events:

- Submit Event: Started upon submission of a form.
- Change Event: Triggered whenever a form element's value changes.
- Events of Focus/Blur: Triggered when an element acquires or loses focus.

3. Load Event:

- Document and Window Events.
- Unload Event: Triggered during the process of unloading a document.
- Resize Event: Started whenever the window of the browser is resized.

4. Network Events:

- Online/Offline Events.
- Asynchronous HTTP request-related events are referred to as Ajax Events.

5. Events of Drag and Drop:

-Dragstart/Dragend Events:Connected to the beginning and conclusion of a drag action.

-A drop's location is indicated by the Drop Event.

6. Media Events:

- Play/Pause Events: Related to the media elements' state of playback.
- Volume Change Event: Started when the media's volume is adjusted.

Developing dynamic and interactive apps requires an understanding of and ability to handle these events. To improve the user experience generally, developers respond to individual events using event handlers and listeners.

Conclusion:

1. Events Significance:

- Events are vital triggers in program execution.
- Include user interactions, system notifications, and external signals.

2. Event Handling:

- Event listeners detect and respond to events.
- Event handlers execute corresponding code.

3. Importance in Web Dev:

- Crucial for responsive, user-friendly web apps.
- Enables real-time updates and dynamic content.

4. Application Across Domains:

- Extends beyond web development.
- Used in GUIs, server-side systems, and data-driven apps.

Understanding events is key for creating applications that respond to user actions, enhancing user experience across various domains.

Comparison and Discussions

Comparison between Declarative Programming and Event-Driven Programming:

1. Readability and Comprehension:

- Declarative: Emphasizes expressing intentions for code that is easily readable and self-explanatory.

- Event-Driven: Offers clarity but requires a solid understanding of event flow and handling.

2. Execution Control: - Declarative: Higher abstraction allows developers to focus on desired outcomes, enhancing predictability.

- Event-Driven: Prioritizes responsiveness through an asynchronous approach, introducing potential challenges in control.

3. Lines of Code:

- Declarative: Involves more lines of expressive code, potentially raising the risk of introducing bugs.

- Event-Driven: Typically concise but may require additional code for effective event handling.

Comparison and Discussions

4. Performance:

- Declarative: May face a performance hit due to increased memory allocation and intermediate function calls.

- Event-Driven: Leverages an asynchronous nature, potentially enhancing performance, especially in resource-intensive tasks.

5. Debugging:

- Declarative: Extended debugging due to higher-level abstractions; larger stack traces may complicate issue identification.

- Event-Driven: Complex debugging due to asynchronous and unpredictable event execution during runtime.

6. Developer Comfort and Learning Curve:

- Declarative: Developers accustomed to imperative paradigms may find it more familiar, but newcomers may face a learning curve.

- Event-Driven: Might present a steeper learning curve, particularly for those unfamil with asynchronous and event-driven concepts.

Comparison and Discussions

7. Separation of Concerns:

- Declarative: May segregate UI or functionality description from driving logic, potentially posing challenges in certain scenarios.

- Event-Driven: Encourages modularity by separating concerns through distinct event handlers.

8. Traceability and Unfolding:

- Declarative: Abstracts away execution details, complicating the tracing of event sequences.

- Event-Driven: Processes events as they occur, ensuring current information, yet potentially making sequence tracing more challenging in complex scenarios.

9. Scalability:

- Declarative: May lack inherent scalability, especially in resource-intensive scenarios.

- Event-Driven: Efficiently utilizes system resources, enabling scalable solutions both vertically and horizontally.

Comparison and Discussions

10. Versatility:

- Declarative: May be more specialized, focusing on specific domains.

- Event-Driven: Demonstrates versatility, finding applications across diverse domains.

11. Complexity:

- Declarative: Generally less intricate, but verbosity might be a concern.

- Event-Driven: Heightened complexity due to the asynchronous nature, demanding precision in synchronization.

12. Dependencies:

- Declarative: Tends to have fewer dependencies on external libraries.

- Event-Driven: May depend on external libraries for event management, introducing complexities in deployment and maintenance.

Listing 3: Declerative code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>.....</style>
  <title>Declarative Example</title>
</head>
<body>
  <button onclick="changeColor()">Change Color</button>
  <div id="content" style="color: blue;">
    This is a declarative example.
  </div>
  <script>
    function changeColor() {
      document.getElementById('content').style.color = 'red';
    }
  </script>
</body>
</html>
```

Listing 4: Event driven code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>....</style>
  <title>Event-Driven Example</title>
</head>
<body>
  <button id="colorButton">Change Color</button>
  <div id="content" style="color: blue;">
  </div>
  <script>
    document.getElementById('colorButton').addEventListener('clic
      document.getElementById('content').style.color = 'green';
    });
  </script>
</body>
</html>
```

Comparison and Discussions

Discussion

-In summary, the choice between Declarative and Event-Driven Programming hinges on factors such as application nature, performance requirements, developer expertise, and the need for modularity and responsiveness.

-Each paradigm possesses distinct strengths and weaknesses, and the appropriateness depends on the specific goals and constraints of the project.

Conclusion

- Consider Project Requirements: Choose based on project-specific needs and goals.

- Declarative Prioritizes: Prioritize declarative programming for readability and higher abstraction.

- Event-Driven Excels: Opt for event-driven programming for enhanced responsiveness and dynamic interactions.

- Thoughtful Decision: Make an informed decision based on project goals and trade-offs.

- Project-Specific: Choose the paradigm that aligns with the specific requirements of the project.

- Informed Adoption: Consider the strengths and weaknesses of each paradigm for an effective adoption strategy.

Biblography

Links for Articles and Blogs:

- https://hazelcast.com/glossary/event-driven-architecture/
- https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/javascript/
- https://app.studysmarter.de/studyset/16476769/summary/70417870
- https://app.studysmarter.de/studyset/16476769/summary/70417870
- https://medium.com/@alexandragrosu03/event-driven-programming-concepts-and-examples-38a985b2a6d6
- https://www.cs.purdue.edu/homes/bxd/java/Applets/wk7.pdf
- https://www.youtube.com/watch?app=desktopv=3PplVtJpLIMt=0
- https:https://dev.to/ruizb/declarative-vs-imperative-4a7l: :text=the
- https://www.composingprograms.com/pages/43-declarative-programming.html: :text=SQL
- https://www.toptal.com/software/declarative-programming: :text=work
- https://www.techtarget.com/searchitoperations/definition/declarative-programming: :text=Declarative
- https://dev.to/ruizb/declarative-vs-imperative-4a7l: :text=the
- https://hazelcast.com/glossary/event-driven-architecture/: :text=Event