Vidya Jyothi Institute of Technology

Academic Year: 2022-2023

Department of Information Technology

Operating Systems

Faculty Name: Dr.A.Obulesu

Teaching Methodology: Cooperative Learning Method

Topic Name: CPU Scheduling Algorithms

Description about Mode:

The process of cooperative learning involves students working together in small groups on a structured activity. The members of the groups learn to work as a team to accomplish a specific goal, to solve a problem, to complete a project, or to develop a product.

Topic Handled:

CPU scheduling algorithms that use a cooperative method are those where processes voluntarily give up control of the CPU, typically after completing their time slice or when they are in a state where they can no longer continue. In such a scheduling method, the running process must explicitly yield the CPU to allow other processes to run. These algorithms are often simpler than preemptive scheduling algorithms and focus on process cooperation.

Some common cpu scheduling algorithms are:

1. First-Come, First-Served (FCFS)

- **Description**: In FCFS, processes are executed in the order they arrive in the ready queue. Once a process starts executing, it runs to completion without interruption, meaning it cooperatively allows other processes to execute only when it finishes.
- **Cooperation**: Processes do not pre-empt each other; they must voluntarily release the CPU after completion. There is no explicit yielding in this case since the process completes its task in one go.
- **Drawback**: If a long process arrives first, it may cause "convoy effect," delaying shorter processes.

2. Shortest Job Next (SJN) / Shortest Job First (SJF)

- **Description**: This scheduling algorithm selects the process with the shortest burst time (or execution time) next. It's a cooperative scheduling method as long as processes complete their execution without pre-emption.
- **Cooperation**: Similar to FCFS, a process that runs executes to completion and voluntarily yields the CPU. The next process chosen is based on its burst time (if known).
- **Drawback**: This algorithm requires knowledge of the execution time of processes in advance, which is often not possible.

3. Round Robin (RR)

- **Description**: Round Robin is a widely used cooperative scheduling algorithm where each process is assigned a fixed time slice (quantum). When a process's quantum expires, it is put at the end of the ready queue, and the CPU is handed over to the next process.
- **Cooperation**: In a cooperative system, the process must voluntarily yield after its quantum expires. If the process completes before its quantum ends, it cooperatively releases the CPU early.
- **Drawback**: The performance may be affected by the size of the time quantum. If the quantum is too large, the system may behave similarly to FCFS. If the quantum is too small, there is frequent context switching, which can reduce performance.

4. Priority Scheduling (Cooperative version)

- **Description**: This algorithm assigns a priority to each process, and the process with the highest priority gets executed next. In a cooperative version, processes voluntarily yield after their execution or when they finish their task.
- **Cooperation**: The process runs to completion or yields voluntarily, and the CPU is given to the next highest-priority process in the queue.
- **Drawback**: Starvation may occur for low-priority processes if higher-priority processes continue to arrive.

Key Characteristics of Cooperative CPU Scheduling:

- **No Pre-emption** Once a process starts running, it is not forcibly stopped by the scheduler. The process must voluntarily give up the CPU.
- **Simple Implementation**: Cooperative scheduling is generally easier to implement than preemptive scheduling since it does not involve complex interrupt handling or context switching.
- **Process Collaboration**: Processes are expected to cooperate, meaning they behave in such a way that they give the CPU back willingly, ensuring fairness in CPU utilization.

Drawbacks:

- **Starvation**: Processes with higher priorities may starve lower-priority processes if they never give up the CPU.
- **Inefficiency**: If a process doesn't yield the CPU properly or runs for an unnecessarily long time, it can cause inefficiency in the overall system.

In summary, cooperative scheduling relies on processes' voluntary actions to release the CPU, making it simpler but also potentially less efficient than preemptive methods.

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION

Academic Year: 2022-2023

DEPARTMENT OF INFORMATION TECHNOLOGY

Design and Analysis of Algorithms

Faculty Name: B.Deepthi

Teaching Learning Methodology: Activity Based Learning

Topic Name: Dynamic Programming

Activity-Based Learning (ABL) is an interactive and engaging approach where students actively participate in their learning process rather than passively receiving information. It encourages problem-solving, critical thinking, and hands-on learning, making concepts more memorable and practical.

Dynamic Programming:

Dynamic Programming (DP) is a powerful technique used in computer science and mathematics to solve problems that can be broken down into simpler subproblems. It is particularly useful in optimization problems, where the solution can be constructed from solutions to smaller subproblems.

Here's an interactive way to learn Dynamic Programming through activities:

Activity 1: Fibonacci sequence (Basic Introduction to DP)

Objective: Introduce the concept of overlapping subproblems and optimal substructure.

Task: Write a program to find the nth number in the Fibonacci sequence using both the recursive approach and the dynamic programming (DP) approach (memoization and tabulation).

Steps:

1. **Recursion without DP:**
   - Write a recursive function to compute Fibonacci numbers.
   - Observe how the same subproblems (like Fibonacci(2)) are computed multiple times.
   - Discuss the inefficiency of this approach.
2. **Memoization (Top-down DP):**
   - Store the results of subproblems in a cache (e.g., an array or dictionary).
   - When you need the result of a previously solved subproblem, you fetch it from the cache instead of recomputing it.
3. **Tabulation (Bottom-up DP):**
   - Build the solution iteratively by solving all subproblems in a bottom-up manner and storing the results.
   - Start from the base cases and compute the Fibonacci sequence up to the nth value.

Activity 2: Coin Change Problem (Optimization Problem)

Objective: Learn how dynamic programming can optimize solutions for problems that require finding the minimum or maximum of a value.

Task: Find the minimum number of coins required to make a given amount, using a set of available denominations.

Steps:

1. **Recursive Approach:**
   - Write a recursive function that explores all possible ways of making the amount by using different coin denominations.
   - Observe the overlapping subproblems, as the same amounts will be computed multiple times.
2. **Top-down DP (Memoization):**
   - Use a memoization approach to store the results of each subproblem.
   - Reuse the stored results to avoid redundant calculations.
3. **Bottom-up DP (Tabulation):**
   - Construct the solution iteratively by starting with an amount of 0 and building up to the target amount using available coins.
   - Create a table where each entry represents the minimum coins required to make that amount.

Activity 3: Knapsack Problem (0/1 Knapsack)

Objective: Demonstrate how DP can be used to solve problems with constraints where you need to make optimal choices.

Task: Given a set of items, each with a weight and value, find the maximum value you can carry in a knapsack of a given capacity.

Steps:

1. **Recursive Approach:**
   - Write a recursive function to explore all possible subsets of items.
   - Track the maximum value that can be obtained for a given capacity.
2. **Top-down DP (Memoization):**
   - Store the results of the subproblems (e.g., maximum value for a specific weight and number of items) in a table.
   - Reuse the results to avoid recalculating.
3. **Bottom-up DP (Tabulation):**
   - Create a table to store the maximum value achievable for each weight up to the knapsack's capacity.
   - Start by considering each item and progressively building the solution based on previously computed results.

Activity 4: Longest Common Subsequence (LCS)

Objective: Teach the importance of optimal substructure and overlapping subproblems in real-world problems like text comparison.

Task: Given two sequences, find the length of the longest subsequence that appears in both sequences.

Steps:

1. **Recursive Approach:**
    - Write a recursive function to compare characters and explore all possible subsequences.
    - Observe how overlapping subproblems occur when comparing the same subsequences multiple times.
2. **Top-down DP (Memoization):**
    - Create a memoization approach where results of subproblems (subsequences) are stored and reused.
3. **Bottom-up DP (Tabulation):**
    - Build a 2D table where each cell represents the longest common subsequence for the first i characters of string A and the first j characters of string B.
    - Use the recurrence relation to fill in the table iteratively.

Activity 5: Matrix Chain Multiplication

Objective: Teach the use of dynamic programming to optimize complex problems involving multiplication of matrices.

Task: Given a sequence of matrices, find the most efficient way to multiply them together. The goal is to minimize the total number of scalar multiplications.

Steps:

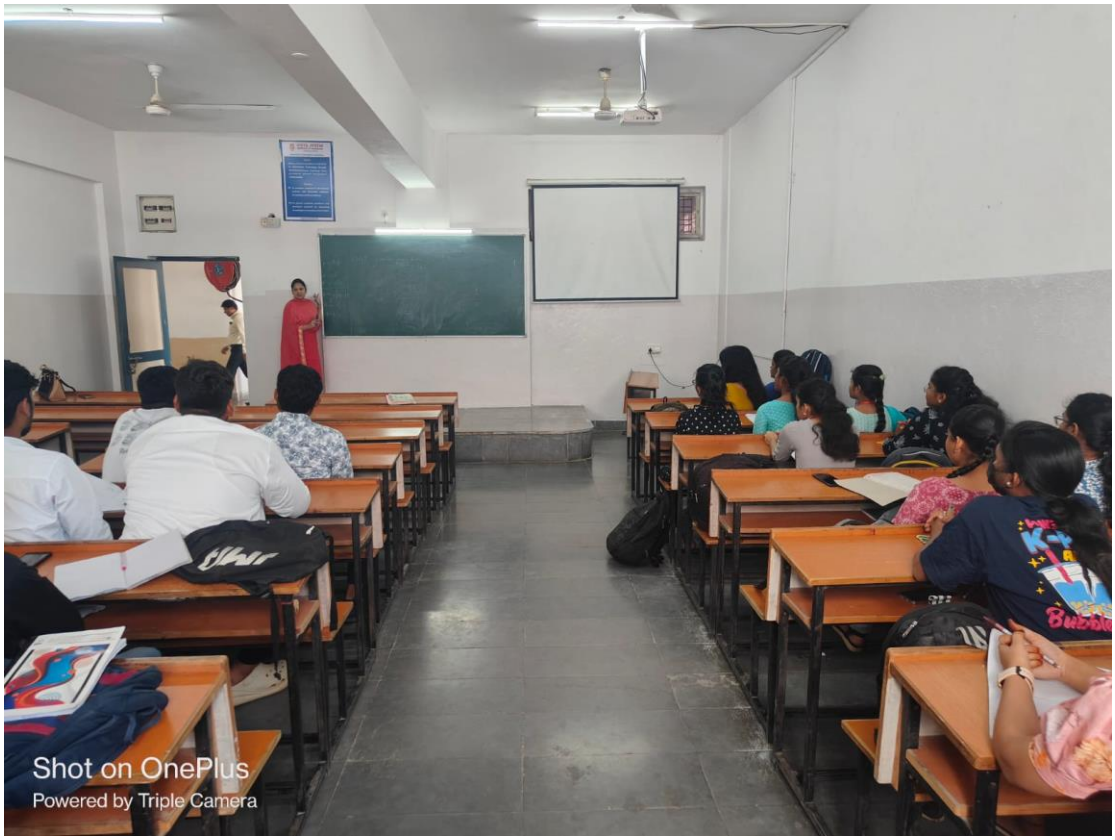
1. **Recursive Approach:**
    - Write a function that recursively tries all possible ways to parenthesize the matrix product.
    - Notice the overlap of subproblems when considering different parenthesizations.
2. **Bottom-up DP (Tabulation):**
    - Construct a table to store the minimum number of scalar multiplications needed for different ranges of matrices.
    - Use the DP approach to build up the solution by considering increasing chain lengths.

Discussion and Reflection:

1. **Optimal Substructure and Overlapping Sub problems:**
    - Discuss how dynamic programming solves problems that have overlapping subproblems and optimal substructure.
    - Have learners reflect on how this concept is applied in the problems above.
2. **Comparing Approaches:**
    - Compare the recursive, memoization, and tabulation methods.
    - Discuss time and space complexities of different methods.

Conclusion:

These activities should help reinforce the core concepts of dynamic programming through hands-on exploration. By solving real-world problems like Fibonacci, coin change, knapsack, and LCS, students will get a deeper understanding of how to apply DP in different scenarios.



Faculty

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION

Academic Year: 2022-2023

Department of Information Technology

Database Management Systems

Faculty Name: Mohd Sirajuddin

Teaching Methodology: Project-Based Learning

Topic: Normalization

Normalization in Database Management Systems (DBMS)

Teaching-Learning Method: Project-Based Learning

Introduction:
Normalization is a process in database design that aims to minimize data redundancy and improve data integrity by organizing data into well-structured tables. It involves applying a series of rules or "normal forms" to break down large tables into smaller, more manageable ones. In a **Project-Based Learning (PBL)** setting, students can learn normalization through hands-on projects that simulate real-world database optimization tasks, enhancing their problem-solving and analytical skills.

Key Concepts of Normalization for PBL

1. First Normal Form (1NF)

- **Description**:
  A table is in 1NF if it has no repeating groups or arrays and each cell contains a single atomic value.
- **Rules**:
  1. Ensure each column contains atomic values.
  2. Eliminate duplicate columns within the same table.
- **Benefits**: Simplifies data structure and eliminates data repetition within rows.
- **Drawbacks**: May lead to a larger number of tables.
- **PBL Application**:
  Students convert a flat-file structure (e.g., customer orders with multiple items in one column) into 1NF by creating a separate row for each item in an order.

2. Second Normal Form (2NF)

- **Description**:
  A table is in 2NF if it is in 1NF and all non-key attributes are fully dependent on the primary key.
- **Rules**:

1. Identify composite primary keys.
2. Remove partial dependencies by separating attributes into new tables.

- **Benefits**: Reduces redundancy caused by partial dependencies.
- **Drawbacks**: Increases complexity by introducing additional tables.
- **PBL Application**: Students normalize a table (e.g., Orders table with redundant customer information) by separating customer details into a new table and linking them with a foreign key.

3. Third Normal Form (3NF)

- **Description**:
  A table is in 3NF if it is in 2NF and all non-key attributes are only dependent on the primary key (eliminating transitive dependencies).
- **Rules**:
  1. Identify attributes indirectly dependent on the primary key.
  2. Move these attributes to a separate table linked by a foreign key.
- **Benefits**: Eliminates redundancy from transitive dependencies, ensuring data consistency.
- **Drawbacks**: Adds more tables, which can make database design more complex.
- **PBL Application**: Students normalize a database containing invoices to separate supplier and product information, ensuring that all non-key attributes relate directly to the invoice ID.

4. Boyce-Codd Normal Form (BCNF)

- **Description**:
  A stricter version of 3NF, BCNF is achieved when every determinant is a candidate key.
- **Rules**:
  1. Identify cases where a non-primary attribute is a determinant.
  2. Restructure the table to ensure all determinants are candidate keys.
- **Benefits**: Ensures a highly normalized database structure with minimal redundancy.
- **Drawbacks**: May result in more complex joins in queries.
- **PBL Application**: Students identify and resolve anomalies in a university database by ensuring that course and instructor relationships conform to BCNF.

5. Higher Normal Forms (4NF, 5NF)

- **Description**:
  - **4NF**: Addresses multi-valued dependencies, ensuring that a table contains no non-trivial multi-valued dependencies.
  - **5NF**: Decomposes tables further to eliminate redundancy caused by join dependencies.

- **Benefits**: Reduces redundancy in highly complex datasets.
- **Drawbacks**: Often not necessary for most practical applications.
- **PBL Application**: Students work on advanced scenarios such as a multi-department employee management system, where they eliminate multi-valued dependencies.

Benefits of Using PBL for Normalization

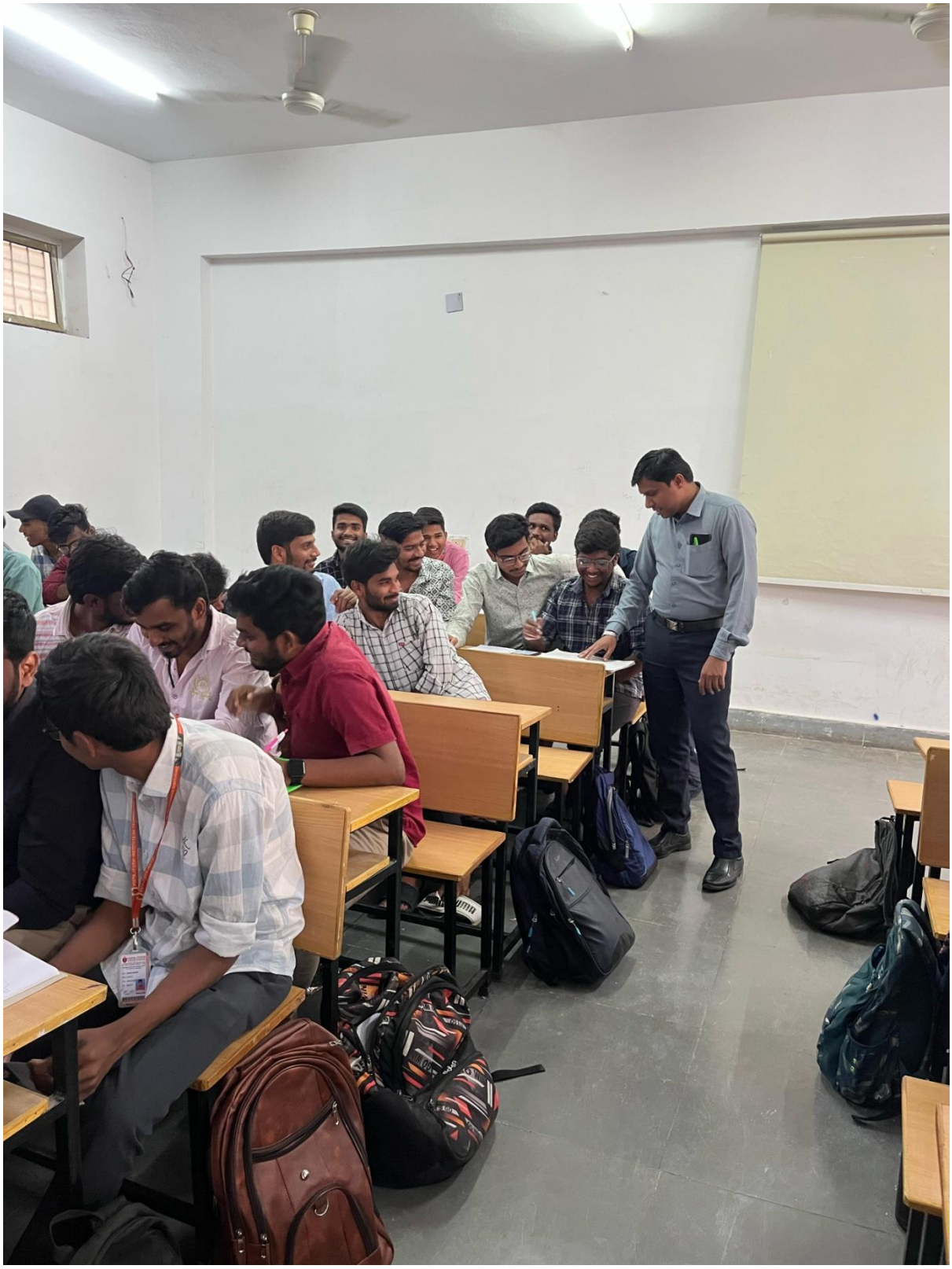
- **Practical Understanding**: Students learn the importance of organizing data effectively in real-world scenarios.
- **Problem-Solving Skills**: Tackling normalization problems enhances analytical thinking.
- **Industry Relevance**: Teaches students techniques commonly used in professional database design.

Challenges of PBL in Normalization

- **Understanding Dependencies**: Identifying and resolving partial or transitive dependencies can be difficult for beginners.
- **Increased Complexity**: Breaking down large datasets into normalized forms may confuse students initially.
- **Time-Consuming**: Normalizing a complex database requires a significant time investment.

Outcome

Through **Project-Based Learning**, students develop a deep understanding of normalization and its practical applications. They gain the ability to design efficient and scalable databases, ensuring data consistency and integrity. These skills prepare them for roles in database administration, software development, and data analysis.



Faculty

VIDYA JYOTHI
INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION

Academic Year: 2022-2023

DEPARTMENT OF INFORMATION TECHNOLOGY

SOFTWARE ENGINEERING

Faculty Name: B. Srinivasulu

Teaching-Learning Method: Project-Based Learning

Topic: Process Models

Project-Based Learning(PBL)in Software Engineering: Process Models

Project-Based Learning (PBL) is a teaching method where students work on real-world projects to apply theoretical knowledge and develop practical skills. In the context of Software Engineering, PBL can be used to engage students in projects that simulate real-world development cycles, focusing on creating software solutions, problem-solving, teamwork and Project Management. To guide these projects, several **process models** are commonly used in the Software Engineering field.

Key Software Engineering Process Models for PBL

1. ***Waterfall Model***

   - **Description**: The Waterfall Model is a linear and sequential approach where each phase must be completed before moving to the next. It's often used in projects where requirements are well-defined and unlikely to change.
   - ***Stages:***

     1. **Requirements Gathering**: Understanding what the software should do.
     2. **System Design**: Planning how the software will work.
     3. **Implementation**: Coding the software.
     4. **Testing**: Verifying that the software works.
     5. **Deployment**: Delivering the software to users.
     6. **Maintenance**: Fixing any issues post-deployment.

   - **Benefits**: Simple to understand and use, especially for smaller projects with

well- defined requirements.

- **Drawbacks**: Lack of flexibility in responding to changing requirements.

PBL Application: In PBL, students could apply the Waterfall model to a software project with a clearly defined set of requirements, such as creating a tool or a small application with predefined functionalities.

2. ***Agile Model***

- **Description**: Agile is an iterative and incremental approach to software development where requirements and solutions evolve through collaboration between self-organizing teams .It emphasizes flexibility, customer collaboration and frequent delivery of small, functional software increments.
- ***Frameworks:***
  - **Scrum**: Organizes development into sprints(short cycles of work),where teams deliver features incrementally.
  - **Kanban**: Focuses on continuous delivery and limiting work in progress (WIP).
- ***Stages:***

  1. Sprint Planning: Defining what will be delivered in a sprint.

  2. Development: Building the product incrementally.

  3. Testing: Continuously testing functionality.

  4. Review: Reviewing the increment with stakeholders.

  5. Retrospective: Analyzing the sprint process to improve performance.
- **Benefits**: Highly flexible and adaptable to changing requirements.
- **Drawbacks**: Requires frequent communication and maybe challenging to implement without a dedicated team.

PBL Application: For a software engineering course, students can use Agile to build an evolving software project over several sprints, iterating on features based on client or stakeholder feedback. They would also experience the roles of Scrum Master, Product Owner and team member.

V-*Model(Verification and Validation)*

- **Description**: The V-Model is an extension of the Waterfall model, where

developmentandtestingarecloselylinked.Eachdevelopmentphasehasa corresponding testing phase.

- ***Stages:***

  1. **RequirementsAnalysis**:Gatheringdetailedrequirements.
  2. **SystemDesign**:Architecturaldesignandhigh-levelplanning.
  3. **DetailedDesign**:Detailedsoftwarecomponentdesign.
  4. **Implementation**:Buildingthe software.
  5. **Unit Testing**: Testingindividual components.
  6. **IntegrationTesting**:Testinginteractionsbetweencomponents.
  7. **SystemTesting**: Validatingthe systemagainst requirements.
  8. **AcceptanceTesting**: Ensuringthesystem meets business needs.

- **Benefits**:Emphasizesthoroughtestingat eachphase.
- **Drawbacks**:SimilartoWaterfall,itlacksflexibilitytochangerequirementsonce development starts.

PBLApplication:InaPBLsetting,studentscouldapplytheV-Modeltocreateasystem where they design, implement, and test software components in parallel, ensuring that each phase of the development cycle includes appropriate validation and verification.

Spiral Model

- **Description**: The Spiral Model combines iterative development with the systematicaspectsoftheWaterfallmodel.Itemphasizesriskmanagement,with each iteration (or spiral) focusing on identifying and mitigating risks.

- ***Stages:***

  1. **Planning**:Determiningobjectivesand alternatives.
  2. **RiskAnalysis**:Identifyingpotentialrisksandhow tomitigatethem.
  3. **Engineering**:Developingthesoftware incrementally.
  4. **Evaluation**:Assessingprogress and revisitingrisk analysis.

- **Benefits**:Veryflexibleand risk-aware,ideal forcomplexprojects.
- **Drawbacks**:Canberesource-intensiveandmayrequireskilledprojectmanagers to manage risks effectively.

PBLApplication:StudentscanusetheSpiralmodelforlarger,morecomplexsoftware projects, where risk management plays a significant role. The team would go through repeatedcyclesofplanning,development,andevaluation,focusingonrisk analysis and mitigation in each cycle.

IncrementalModel

- **Description**: The Incremental Model involves building the software in small, manageable parts or increments, with each part developed and delivered separately.Thefullfunctionalityiscompletedonceallincrementsareintegrated.
- ***Stages:***

  1. Planning:High-levelrequirementsare defined.

  2. IncrementalDevelopment:Eachincrementis developedanddeployed.

  3. IntegrationandTesting:Aftereachincrement,integrationandtesting occur.

  4. Deployment:Thesoftwareisreleasedafterallincrementsare completed.
- **Benefits**:Providesfunctionalsoftwareearlyonandallowsforiterative improvements.
- **Drawbacks**:Requirescarefulplanningof theincrementstoavoidmisalignment.

PBL Application: In a PBL context, students can develop a software project incrementally,whereeachincrementaddressesasubsetoftheoverallrequirements.As the project progresses, the software becomes more complete and functional.

Benefits of Using PBL in Software Engineering

- ✓ **Hands-on Experience**: Students gain direct experience with software development, working with real-world tools and methods.
- ✓ **Problem-Solving Skills**: Working on projects helps students tackle problems and find solutions in a practical context.

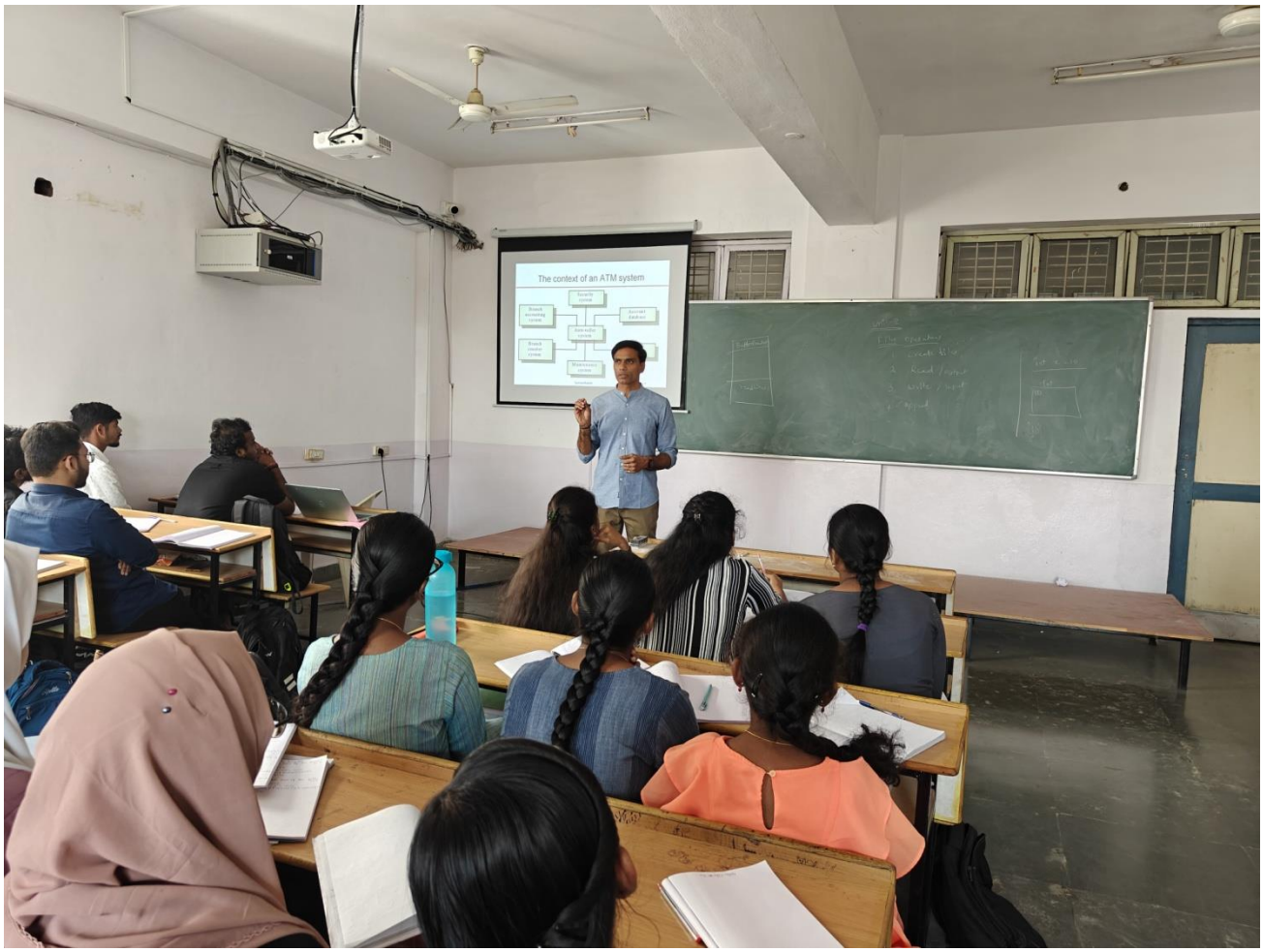
- ✓ **Team work and Communication**:PBL fosters collaboration,an essential skill in the software industry.
- ✓ **Industry Relevance**: Students are exposed to various process models and development practices used in the software industry.

Challenges of PBL in Software Engineering

- **TimeConstraints**:Projectsmayrequiresignificanttimeinvestment,whichcanbe challenging for students with other commitments.
- **Complexity of Real-World Projects**: Real-world software development can be unpredictable,makingithardertosimulateinacontrolledclassroomenvironment.
- **CoordinationandCommunication**:Managingteamdynamicsandensuringsmooth communication can be difficult, especially in large teams.

Outcome

Project-Based Learning in Software Engineering, when paired with effective process models, provides students with invaluable experience and practical skills. By working on real-world projects and applying these process models, students can better understand the challenges and dynamics of software development, preparing them for careers in the field.



Faculty

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION

Academic Year: 2022-2023

DEPARTMENT OF INFORMATION TECHNOLOGY

Object Oriented Programming through JAVA

Faculty Name: B. Eswar Babu

Teaching Learning Methodology: Demonstration Based Learning

Topic: Object-Oriented Concepts in Java

Demonstrative-based learning is a teaching method in which the instructor demonstrates a process, concept, or technique step by step, allowing students to learn through observation and practice. In the context of **searching and sorting algorithms**, demonstrative-based learning helps students understand how these algorithms work, why they are used, and their real-world applications.

A demonstration-based learning plan for teaching **Object-Oriented Concepts in Java**:

Objective

Engage students in learning core Object-Oriented Programming (OOP) concepts using Java through hands-on demonstrations and practical examples.

Session Plan

1. Introduction to OOP (1 hour)

- **Concepts Covered**: Object, Class, Encapsulation, Abstraction, Polymorphism, Inheritance.
- **Demonstration**:
    1. Create a simple Car class with attributes (brand, model, price) and a method displayDetails().
    2. Instantiate objects and call methods.

2. Classes and Objects (1 hour)

- **Concepts Covered**: Defining a class, creating objects, accessing attributes and methods.
- **Demonstration**:
    1. Develop a BankAccount class with methods for deposit(), withdraw(), and displayBalance().
    2. Show object creation and method invocation in the main method.

3. Encapsulation and Abstraction (1 hour)

- **Concepts Covered**: Use of private access modifier, getter, and setter methods, abstraction using interfaces.
- **Demonstration**:
    1. Create a Student class with private attributes name and rollNo. Use getter and setter methods.
    2. Introduce an interface Shape with methods calculateArea() and calculatePerimeter(). Implement it in Circle and Rectangle classes.

4. Inheritance (1 hour)

- **Concepts Covered**: Superclass, Subclass, super keyword, method overriding.
- **Demonstration**:
    1. Develop a superclass Animal with methods makeSound() and subclass Dog overriding it.
    2. Use super to call the parent class constructor or method.

5. Polymorphism (1 hour)

- **Concepts Covered**: Method overloading, method overriding, dynamic method dispatch.
- **Demonstration**:
    1. Create a Calculator class with overloaded methods for add(int, int) and add(double, double).
    2. Demonstrate dynamic method dispatch with an example of Shape superclass and Circle, Square subclasses.

6. Working with Constructors (1 hour)

- **Concepts Covered**: Default constructor, parameterized constructor, constructor overloading.
- **Demonstration**:
  Create a Book class with attributes like title, author, and price. Implement multiple constructors for initialization.

7. Exception Handling (1 hour)

- **Concepts Covered**: Try-catch block, finally, custom exceptions.
- **Demonstration**:
    1. Create a program that performs division and handles ArithmeticException.
    2. Develop a custom exception class InvalidAgeException and use it in an ageValidation() method.

8. Real-Time Demonstration Project (2 hours)

- **Mini-Project**:
  Develop a Library Management System covering all the concepts:
    - Class: Book, Member, Librarian.
    - Encapsulation: Private attributes with getter and setter methods.
    - Inheritance: Common attributes and methods in a base class Person.

- Polymorphism: Overloaded methods for searchBook() by title, author, or genre.
- Exception Handling: Handle errors like invalid member ID.

9. Wrap-Up and Q&A (1 hour)

- Review key concepts through coding challenges and quizzes.
- Allow students to clarify doubts and demonstrate their own examples.

Tools and Platforms

- IDE: IntelliJ IDEA or Eclipse.
- Live coding platform: Codetantra or an equivalent for interactive learning.
- Screen sharing for step-by-step explanations.

Assessment

- Assign tasks to implement OOP concepts. For example:
  - Create a Vehicle hierarchy with attributes and methods.
  - Develop a Banking system with exception handling and polymorphism.

Demonstrations for Object-Oriented Concepts in Java

1. Introduction to OOP

Task Description: Create a simple `Car` class with attributes (`brand`, `model`, `price`) and a method `displayDetails()`. Demonstrate object creation and method invocation.

Code Example:

```
class Car {
   String brand;
   String model;
   double price;

   void displayDetails() {
      System.out.println("Brand: " + brand);
      System.out.println("Model: " + model);
      System.out.println("Price: " + price);
   }
}

public class Main {
   public static void main(String[] args) {
      Car car1 = new Car();
      car1.brand = "Toyota";
      car1.model = "Corolla";
      car1.price = 20000;
      car1.displayDetails();
   }
}
```

2. Classes and Objects

Task Description: Develop a `BankAccount` class with methods for `deposit()`, `withdraw()`, and `displayBalance()`. Demonstrate creating an object and invoking its methods.

Code Example:

```
class BankAccount {
  private double balance;

  void deposit(double amount) {
    balance += amount;
    System.out.println("Deposited: " + amount);
  }

  void withdraw(double amount) {
    if (balance >= amount) {
      balance -= amount;
      System.out.println("Withdrawn: " + amount);
    } else {
      System.out.println("Insufficient funds.");
    }
  }

  void displayBalance() {
    System.out.println("Balance: " + balance);
  }
}

public class Main {
  public static void main(String[] args) {
    BankAccount account = new BankAccount();
    account.deposit(500);
    account.withdraw(200);
    account.displayBalance();
  }
}
```

3. Encapsulation and Abstraction

Task Description: Create a `Student` class with private attributes `name` and `rollNo` and use getter and setter methods. Demonstrate abstraction by creating an interface `Shape` and implementing it in `Circle` and `Rectangle` classes.

Code Example for Encapsulation:

```
class Student {
  private String name;
  private int rollNo;

  public String getName() {
```

```
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getRollNo() {
    return rollNo;
  }

  public void setRollNo(int rollNo) {
    this.rollNo = rollNo;
  }
}
```

Code Example for Abstraction:

```
interface Shape {
  double calculateArea();
  double calculatePerimeter();
}

class Circle implements Shape {
  private double radius;

  Circle(double radius) {
    this.radius = radius;
  }

  public double calculateArea() {
    return Math.PI * radius * radius;
  }

  public double calculatePerimeter() {
    return 2 * Math.PI * radius;
  }}
```

4. Inheritance

Task Description: Create a superclass `Animal` and a subclass `Dog`. Demonstrate method overriding and the use of the `super` keyword.

Code Example:

```
class Animal {
  void makeSound() {
    System.out.println("Animal makes a sound");
  }
}
```

```
class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Dog barks");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
    }
}
```

5. Polymorphism

Task Description: Demonstrate method overloading by creating a `Calculator` class. Show method overriding through dynamic method dispatch.

Code Example:

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println(calc.add(2, 3));
        System.out.println(calc.add(2.5, 3.5));
    }
}
```

6. Working with Constructors

Task Description: Create a `Book` class with attributes like `title`, `author`, and `price`. Demonstrate the use of default and parameterized constructors.

Code Example:

```
class Book {
    String title;
    String author;
    double price;
```

```
    Book() {
        System.out.println("Default Constructor");
    }

    Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    void displayDetails() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Price: " + price);
    }
}
```

7. Exception Handling

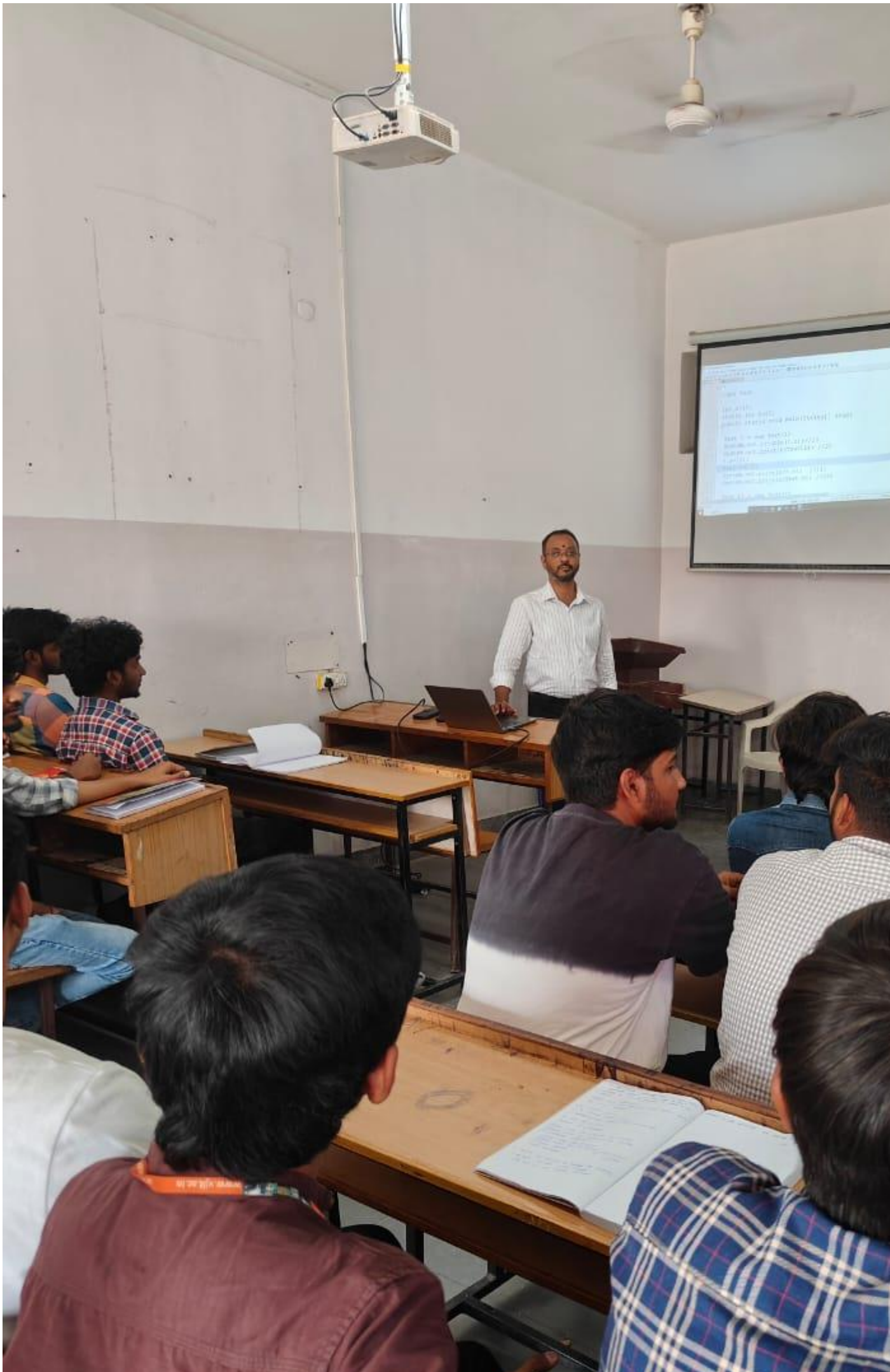
Task Description: Create a program that handles exceptions. Develop a custom exception class `InvalidAgeException` and use it in a method `validateAge()`.

Code Example:

```
class InvalidAgeException extends Exception {
    InvalidAgeException(String message) {
        super(message);
    }
}

public class Main {
    static void validateAge(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age must be 18 or above.");
        } else {
            System.out.println("Valid age.");
        }
    }

    public static void main(String[] args) {
        try {
            validateAge(16);
        } catch (InvalidAgeException e) {
            System.out.println("Exception: " + e.getMessage());
        } }}
```



VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION

Academic Year: 2022-2023

DEPARTMENT OF INFORMATION TECHNOLOGY

Computer Networks

Faculty Name: Laxmi Hugar

Teaching-Learning Method: ICT based learning

Topic Name: OSI Models

ICT based learning in Computer Networks: OSI Models

Information and Communication Technology-based learning, refers to the integration of digital tools and resources to enhance teaching and learning processes. ICT-based learning on the **OSI Model of Networking** involves using digital tools and resources to teach and explore the layers of the Open Systems Interconnection model in an interactive and practical way

Teaching Approaches Using ICT

1. **Multimedia Resources**:
   - Use **animations and videos** to visually demonstrate the seven OSI layers and their roles.
   - Example: Show how data flows from the Application layer to the Physical layer during transmission.
2. **Simulations and Tools**:
   - Employ tools like **Cisco Packet Tracer**, **GNS3**, or **Wireshark** to simulate network environments and analyze layer-specific operations.
   - Example: Simulate a TCP handshake (Transport layer) or observe packet encapsulation and de-encapsulation.
3. **Gamification**:
   - Integrate games and quizzes that match protocols (e.g., HTTP, TCP, IP) to their respective OSI layers.
   - Example: A game where learners drag and drop functionalities (e.g., "Ensures reliable transmission") to the correct OSI layer.
4. **Virtual Labs**:
   - Conduct **virtual experiments** where students set up networks, capture packets, and explore the functioning of protocols at each layer.
   - Example: Use Wireshark to analyze Application layer headers and data.
5. **Collaborative Learning Platforms**:
   - Use platforms like **Google Classroom** or **Microsoft Teams** for group activities, discussions, and sharing resources on OSI model concepts.
   - Example: Assign group projects to design a network architecture and explain how OSI layers interact.

Learning Outcomes

Through ICT-based learning, students will:

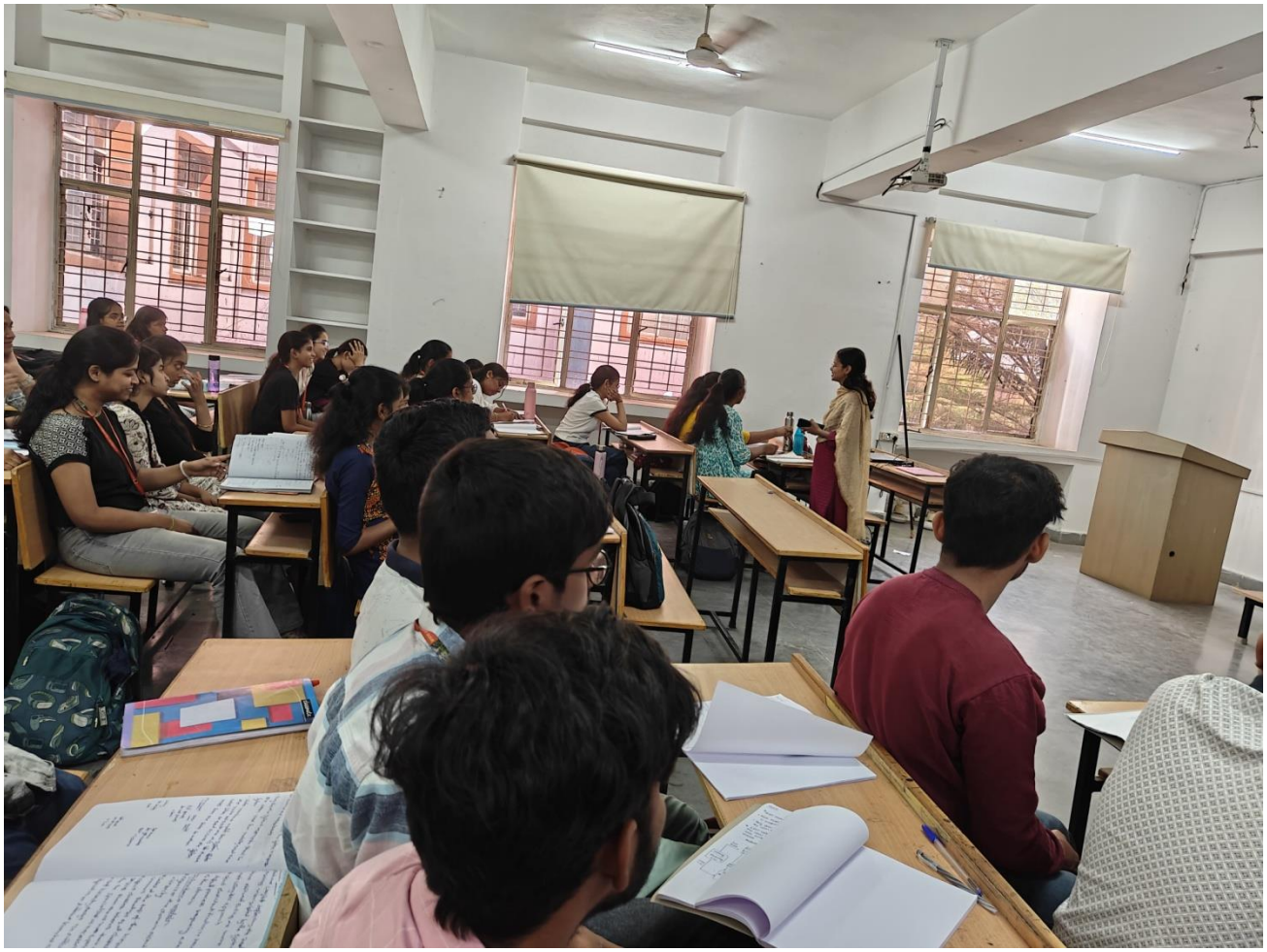
- Understand the **roles and responsibilities** of each OSI layer.
- Gain hands-on experience with **network simulation tools**.
- Develop the ability to **analyze real-world networking scenarios**, bridging theory and practice.

Example Activity

Objective: Explore the Network Layer (OSI Layer 3).
Task:

- Use a network simulation tool like Packet Tracer to set up a network with routers and switches.
- Configure IP addressing and routing protocols.
- Observe how data packets are routed across the network.

By integrating ICT tools into the teaching of the OSI model, students can achieve a deeper and more practical understanding of computer networking concepts.



Faculty

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION

Academic Year: 2022-2023

DEPARTMENT OF INFORMATION TECHNOLOGY

Big Data Analytics

Faculty Name: J.Bramaramba

Teaching Learning Methodology: Inquiry-Based Learning

Topic: Hadoop Distributed File System

Inquiry-Based Learning Methodology

Inquiry-based learning is a teaching strategy that emphasizes the importance of students actively engaging with questions, exploring content through research and experimentation, and drawing conclusions from their findings. When applied to the Hadoop Distributed File System (HDFS), inquiry-based learning encourages students to delve into the architecture, components, and functionalities of HDFS by solving problems, conducting experiments, and collaborating on projects.

Hadoop Distributed File System (HDFS)

Introduction: HDFS is the primary storage system of Hadoop. It is designed to store vast amounts of data reliably and to stream those datasets to user applications at high bandwidth. Understanding HDFS is critical for working with Big Data technologies.

Key Components of HDFS

1. **NameNode**
    - **Definition:** The NameNode is the master server that manages the metadata and directory structure of HDFS. It maintains information about file locations and ensures data integrity.
    - **Inquiry-Based Activity:**
        - **Research Task:** Investigate the role of the NameNode in ensuring fault tolerance. How does it handle failures?
        - **Experiment:** Simulate a NameNode failure in a test Hadoop environment and explore the system's recovery mechanisms.

        - **Discussion:** Why is the NameNode critical, and how does its architecture influence the scalability of HDFS?
2. **DataNode**
    - **Definition:** DataNodes are the worker nodes that store the actual data in HDFS. They communicate with the NameNode to perform storage operations.
    - **Inquiry-Based Activity:**
        - **Research Task:** What strategies does the DataNode use for block replication? How does replication ensure data reliability?
        - **Experiment:** Configure a DataNode and explore how block storage and replication work in HDFS.
        - **Discussion:** How do DataNodes contribute to the performance and fault tolerance of HDFS?
3. **Block Storage**
    - **Definition:** Data in HDFS is divided into blocks, which are distributed across multiple nodes.
    - **Inquiry-Based Activity:**
        - **Research Task:** What are the advantages of using block-based storage compared to traditional file systems?
        - **Experiment:** Test the performance of different block sizes (e.g., 64MB, 128MB) in a sample HDFS cluster.
        - **Discussion:** How does block storage impact scalability and performance?
4. **Replication Factor**
    - **Definition:** Replication ensures that multiple copies of each block are stored across different nodes to safeguard against data loss.
    - **Inquiry-Based Activity:**
        - **Research Task:** Analyze the impact of varying replication factors on storage efficiency and fault tolerance.
        - **Experiment:** Configure different replication factors and observe their effect on data availability.
        - **Discussion:** What trade-offs exist between fault tolerance and storage overhead?

Inquiry-Based Learning Activities

1. **Data Ingestion and Storage**

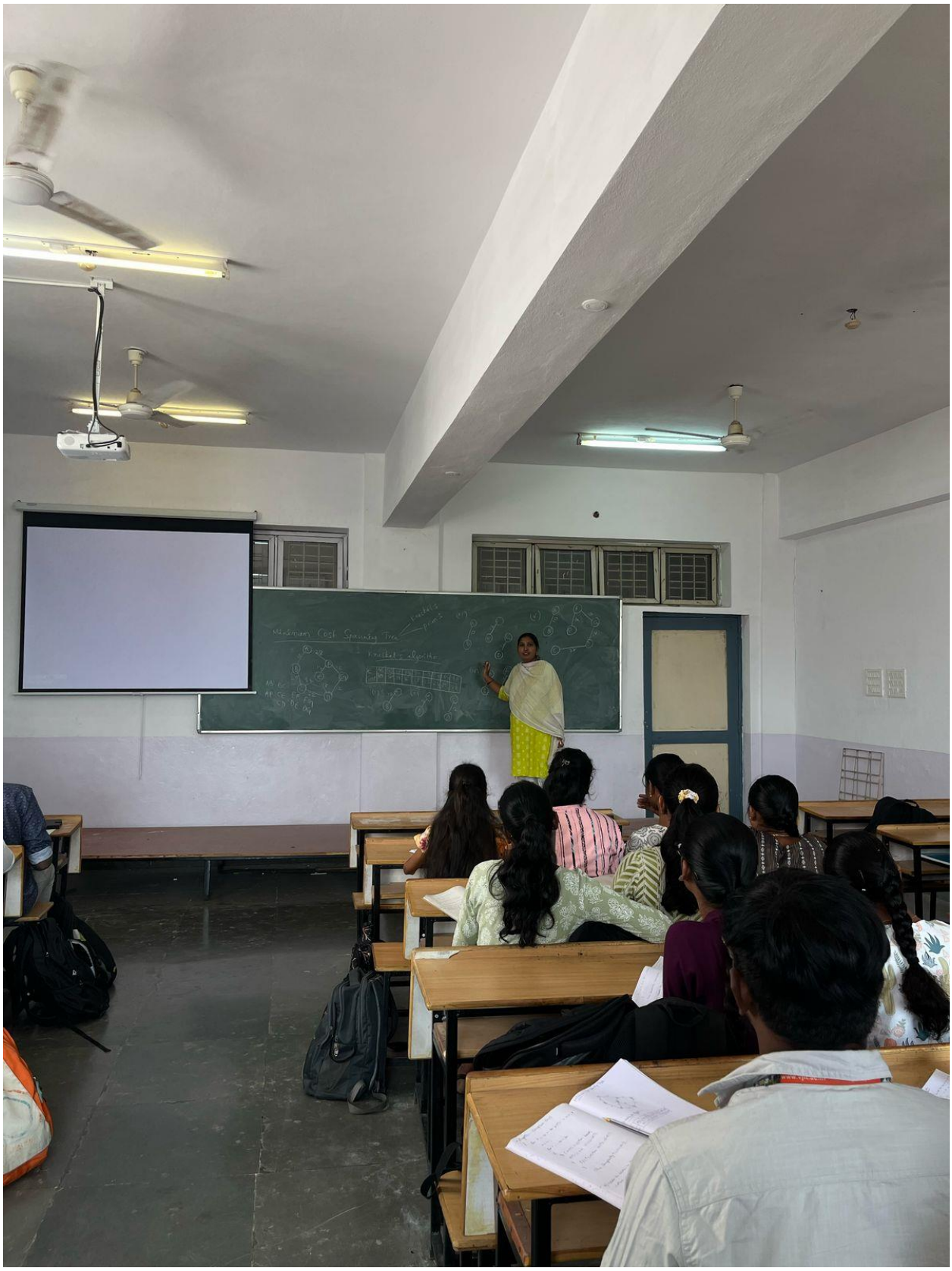
- **Scenario:** Students are tasked with ingesting a large dataset into HDFS and analyzing its storage distribution.
- **Steps:**
  - Research the file formats supported by HDFS.
  - Experiment with tools like Hadoop File System Shell commands to ingest and retrieve data.
  - Discuss how the file size and format impact storage and retrieval performance.

2. **Fault Tolerance Investigation**
   - **Scenario:** Simulate a scenario where a DataNode or NameNode fails.
   - **Steps:**
     - Research how HDFS detects and recovers from node failures.
     - Conduct experiments to observe system behavior during failures and recovery.
     - Discuss the limitations and strengths of HDFS fault tolerance mechanisms.
3. **Performance Tuning**
   - **Scenario:** Optimize HDFS performance for a given dataset and workload.
   - **Steps:**
     - Research configuration parameters such as block size and replication factor.
     - Experiment with tuning these parameters in a test cluster.
     - Discuss the observed trade-offs between throughput, storage efficiency, and fault tolerance.
4. **Real-World Application Design**
   - **Scenario:** Design a scalable storage system for an organization's big data use case.
   - **Steps:**
     - Identify the storage and performance requirements.
     - Collaboratively design a solution using HDFS features.
     - Present findings and justify design decisions.

Conclusion

Understanding HDFS is crucial for managing and analyzing large datasets in Big Data applications. Inquiry-based learning encourages students to explore HDFS through research, experimentation, and collaboration, fostering deeper comprehension and critical thinking

skills. These activities provide practical insights into the architecture and functionalities of HDFS, preparing students for real-world challenges in Big Data environments.



Faculty

Vidya Jyothi Institute of Technology

Academic Year: 2022-2023

Department of Information Technology

Fundamentals of Cyber Security

Faculty Name: Padma Priya J

Teaching Methodology: Interactive Learning

Topic: Tools and methods used in cyber crime

Description about Mode:

Interactive learning is an engaging educational approach that encourages active participation from students through discussions, hands-on activities, and technology. It promotes collaboration, critical thinking, and problem-solving by involving learners directly in the learning process, making it more dynamic and effective in retaining knowledge and skills.

Topic Handled:

Tools and methods used in cyber crime

- Cybercriminals use various tools and methods to commit illegal activities online. Common tools include malware (viruses, ransomware, spyware) that infiltrate systems to steal data or hold it hostage.
- Phishing attacks trick individuals into revealing personal information through fraudulent emails or websites.
- Distributed Denial of Service (DDoS) attacks overload websites, causing them to crash.
- Keyloggers capture keystrokes, enabling identity theft.
- Cybercriminals also exploit vulnerabilities in software and networks, using hacking tools like exploit kits to gain unauthorized access.
- Social engineering, such as pretexting and baiting, manipulates individuals into divulging sensitive information.

These techniques undermine security and privacy across digital platforms.

Outcome of teaching mode:

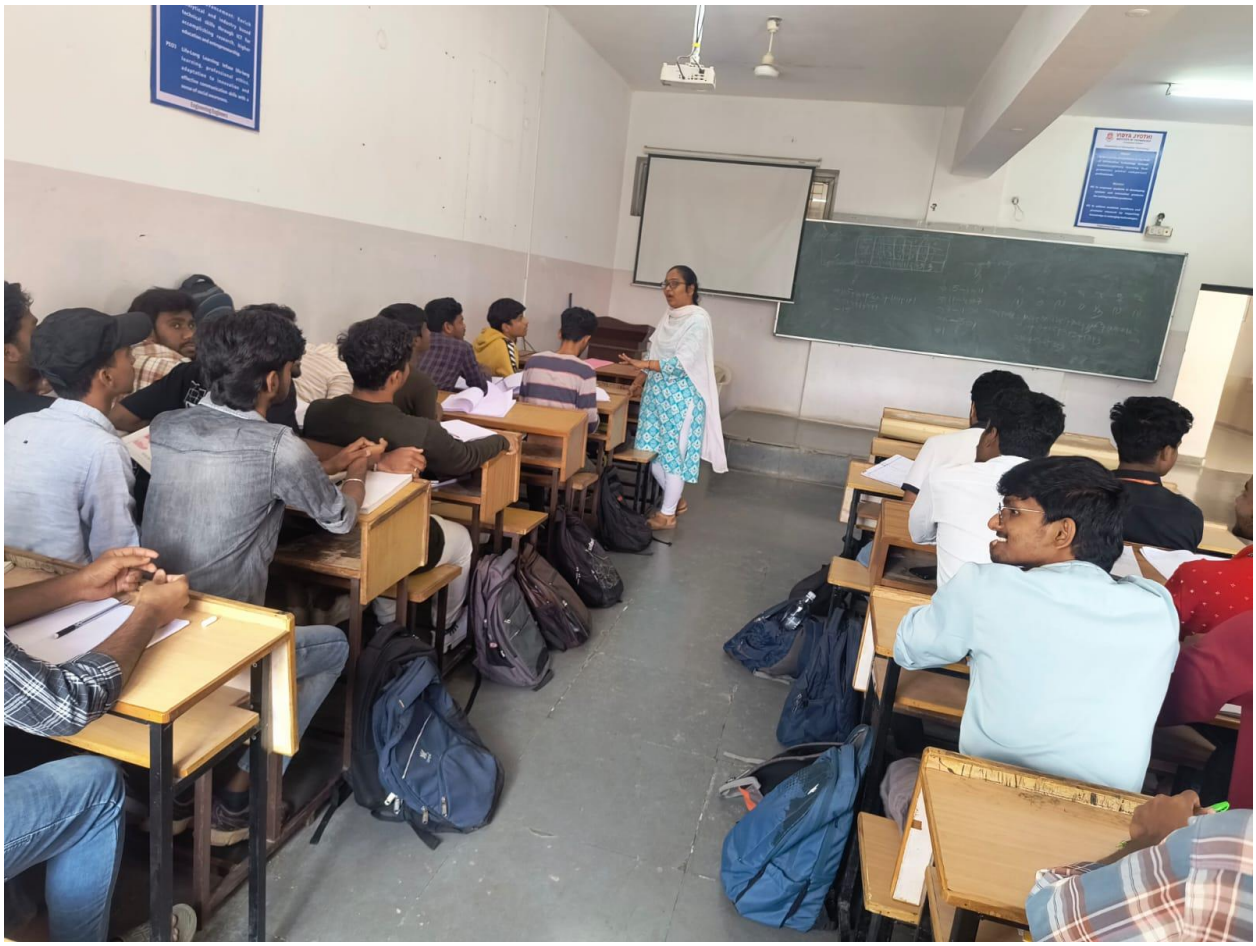
Teaching about the tools and methods used in cybercrime equips students with essential knowledge to recognize, prevent, and respond to digital threats. By

understanding the various techniques employed by cybercriminals, such as malware, phishing, and DDoS attacks, learners become more aware of the risks associated with online activities. This awareness fosters a proactive approach to cybersecurity, enabling students to adopt safer online practices and protect personal and organizational data.

Moreover, teaching these methods encourages critical thinking about ethical implications and the importance of digital responsibility. Students are also better prepared to identify vulnerabilities in systems and use defensive tools to counteract cyber threats. Additionally, the curriculum can inspire interest in careers related to cybersecurity, as it highlights the growing demand for professionals in this field.

Conclusion

Overall, teaching about cybercrime tools and methods not only informs students but also empowers them to contribute to a safer digital environment.



Faculty

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION

Academic Year: 2022-2023

Department of Information Technology

Mathematical Foundations of Computer Science (MFCS)

Faculty Name: M. Keerthi

Teaching Learning Methodology: Interactive Learning.

Topic Name: Combinations

Interactive learning is a student-centered teaching strategy where learners actively engage in discussions, problem-solving, and hands-on activities. When applied to **Combinatorics**, specifically combinations, this method fosters critical thinking and collaborative skills while solidifying theoretical understanding through real-world applications.

Concept: Combinations

Combinations refer to the selection of items from a larger set where the order of selection does not matter. The number of ways to choose items from a set of items is given by:

Interactive Learning Approach

1. Concept Introduction: Visual Learning

- **Use Real-Life Examples:** Begin with examples like selecting members for a committee, choosing flavors of ice cream, or forming teams.
- **Visual Aids:** Use Venn diagrams or graphical representations to explain the difference between permutations and combinations.

2. Group Activity: Exploring Formulas

- **Activity:** Divide students into small groups and assign each group to compute combinations for different values of and .
    - Example Problems:
        - Find (choosing 2 items from 5).
        - Compute .
    - **Discussion Questions:**
        - How does change if or ?
        - What is the symmetry property of combinations? (i.e.,)

3. Hands-On Activity: Combinatorics in Action

- **Material:** Provide decks of cards, colored balls, or coins.
- **Task:**

    - Ask students to calculate how many ways they can pick 3 red balls from a bag containing 5 red and 4 blue balls.
    - Discuss how combinations differ when the selection involves conditions, e.g., "at least one blue ball."

4. Collaborative Learning: Real-World Scenarios

- **Scenario 1:** "A company wants to form a team of 3 employees from a group of 7. How many ways can the team be formed?"
    - Encourage students to discuss and solve in pairs.
- **Scenario 2:** "In a lottery, a player picks 6 numbers out of 49. How many possible combinations are there?"
    - Use calculators or software tools to simplify computations.

5. Problem-Solving Challenge: Gamified Learning

- **Quiz Game:** Organize a rapid-fire round where students solve combination problems under a time limit.
    - Example Questions:
        - "How many ways can you select 2 leaders from 10 students?"
        - "What is the value of ?"
- Award points for correct answers and explanations.

Assessment and Feedback

1. Formative Assessment:

- **Quiz:** Conduct a short quiz after the session to test comprehension.
    - Example: "A basket contains 4 apples, 3 oranges, and 5 bananas. In how many ways can 3 fruits be selected?"
- **Group Presentations:** Have groups present their solutions to assigned problems, explaining their reasoning.

2. Summative Assessment:

- Assign worksheets with real-world problems and encourage students to discuss solutions in pairs.

Collaborative Project

- **Design Challenge:**
    - Ask students to design a "combinatorics-based" game where players calculate combinations to advance levels.
- **Presentation:** Each group presents their game logic and explains the role of combinations in their design.

Conclusion

By integrating interactive and collaborative learning strategies into teaching combinations, students not only master the mathematical techniques but also develop teamwork, problem-solving, and critical-thinking skills. This approach transforms abstract concepts into engaging, hands-on experiences that resonate beyond the classroom.