Name of Student: Pushkar Sane		
Roll Number: 45		**Tutorial Number: 6**
Title of Tutorial: UML Diagrams (Class Diagram)		
DOP: 25-09-2023		**DOS: 26-09-2023**
CO Mapped: **CO1, CO2, CO3**	**PO Mapped:** **PO1, PO4, PO6**	**Signature:**

Tutorial No. 6

Aim: UML Diagrams (Class Diagram)

Description:

- **Introduction to UML Activity Diagrams**

  What is UML?

  Unified Modelling Language (UML) is a standardized modelling language used in software engineering for visualizing, specifying, constructing, and documenting the artifacts of software systems. UML provides various types of diagrams to represent different aspects of a system, and one of the fundamental diagrams is the Class Diagram.

- **Purpose of Class Diagrams**

  Class Diagrams serve as a foundational tool for modelling the static structure of a system. They capture the structure and relationships among classes, which are the blueprint for objects in object-oriented programming. Class Diagrams are used to design and document software systems, making them a critical component in the software development lifecycle.

- **Benefits of Class Diagram**

  Class Diagrams offer several advantages in system design and development:

  1. **Clarity**: They provide a clear visual representation of the system's structure and relationships among classes.
  2. **Communication:** They facilitate communication between developers, designers, and stakeholders by offering a common visual language.
  3. **Design:** They serve as a blueprint for system design, helping developers understand the relationships and attributes of classes.
  4. **Documentation:** They document the static structure of the system, aiding in maintenance and future development.
  5. **Code Generation:** They can be used as a basis for code generation in some modelling tools.

- **Elements of Class Diagram:**

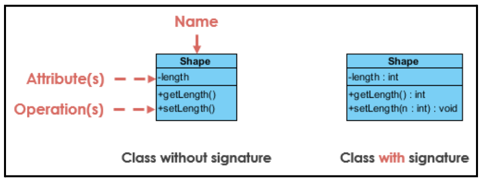
  A Class Diagram consists of several key elements:

  1. **Classes:** Classes are the primary building blocks of a Class Diagram. They represent objects or concepts in the system and define the attributes and operations associated with them.
  2. **Attributes:** Attributes are properties or characteristics of a class. They represent the data that an object of the class can hold. Attributes are typically represented in the form of name: data type.
  3. **Operations:** Operations are functions or methods associated with a class. They define the behavior or functionality of the class. Operations are usually represented with a name followed by parameters and a return type.
  4. **Relationships:** Relationships represent the associations and connections between classes. There are several types of relationships in Class Diagrams, including association, aggregation, composition, inheritance, and dependency.
  5. **Multiplicity Notations:** Multiplicity notations indicate how many instances of one class are associated with an instance of another class in an association relationship. They are represented using numbers, such as 1, 0..1, or 0...

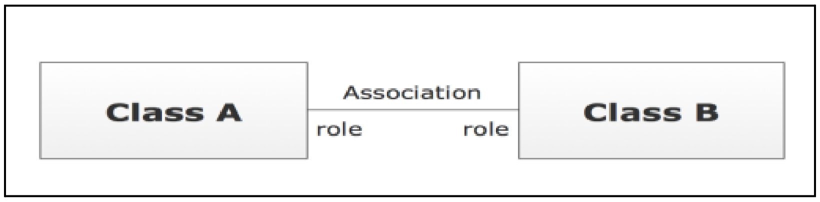
- **Class Diagram Notations**

  Understanding the notations used in Class Diagrams is crucial for creating and interpreting them effectively.

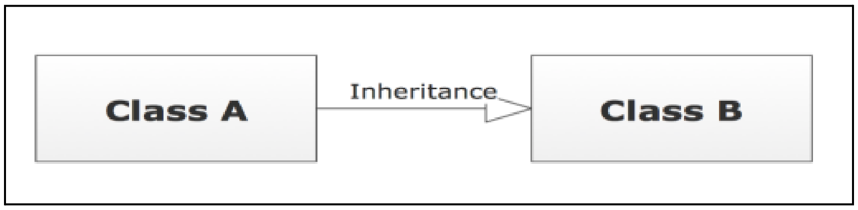
  1. **Class Notation:** Classes are typically represented as rectangles with three compartments, the top compartment contains the class name, the middle compartment lists attributes, and the bottom compartment lists operations.



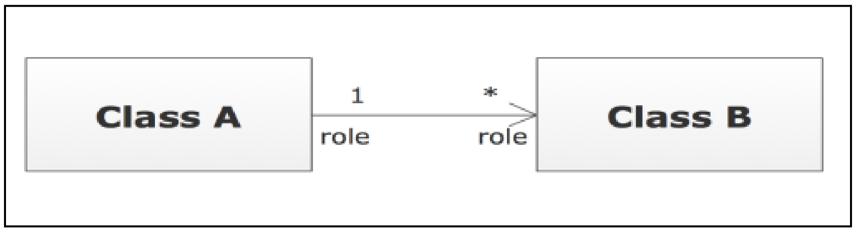
2. **Association Notation:** Associations between classes are depicted as lines connecting them. Multiplicity notations may be added to indicate the number of instances associated.



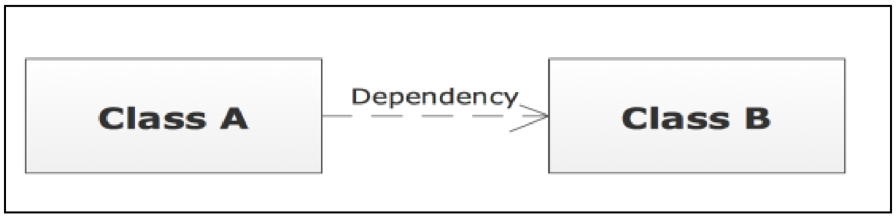
3. **Inheritance Notation:** Inheritance relationships, which indicate that one class inherits properties and behaviors from another, are represented with a line that has an arrowhead pointing to the superclass.



4. **Multiplicity Notation:** Multiplicity notations are shown near the end of an association line and indicate how many instances of one class are associated with instances of another class.



5. **Dependency Notation:** Dependencies between classes, where one class relies on another, are represented by a dashed line with an arrow pointing to the dependent class.



- **Creating Class Diagrams**

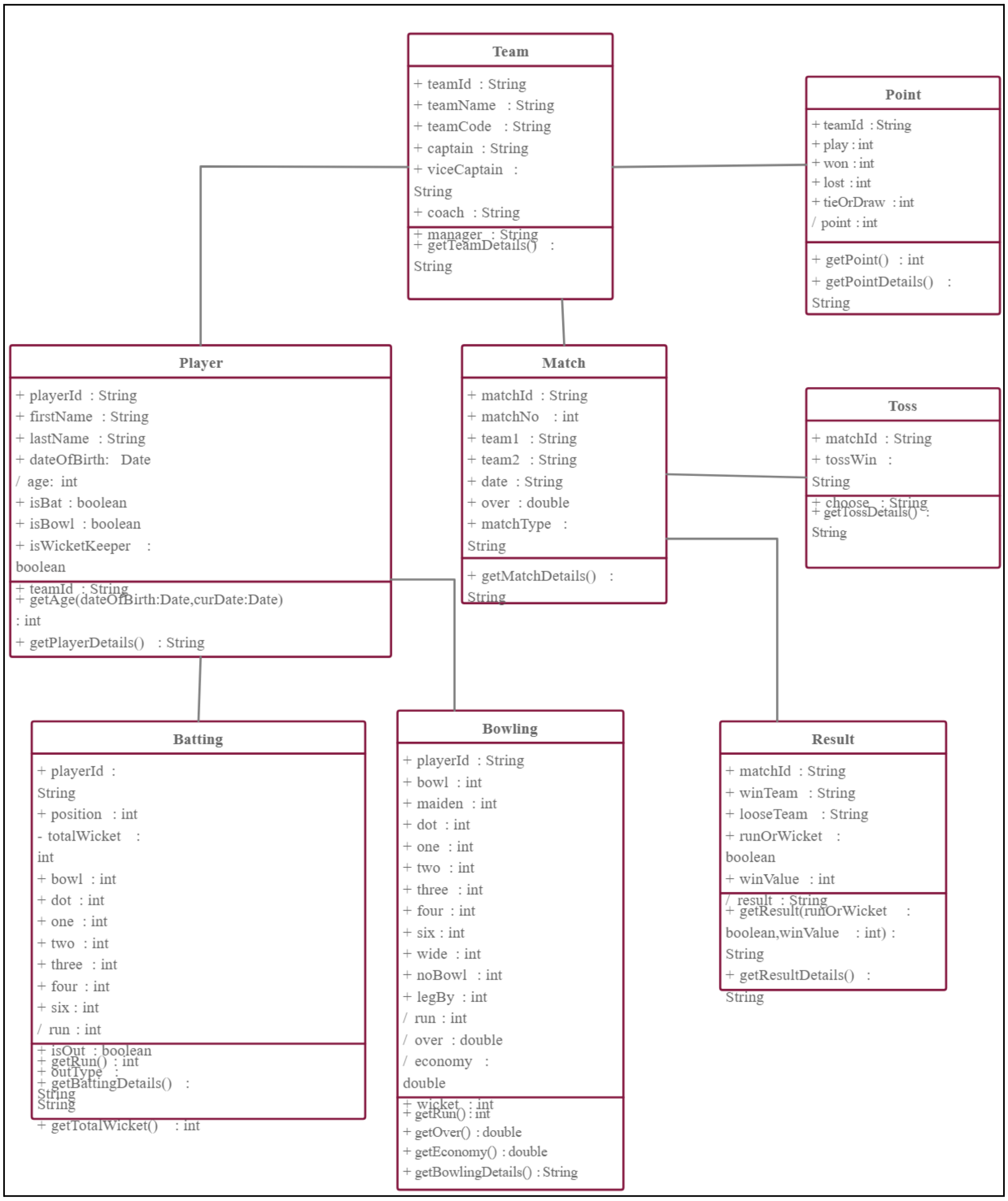
  Creating a Class Diagram involves several steps:

  1. **Identify Classes:** Identify the classes that are essential for modelling the system. These classes represent the main components of the system.
  2. **Define Attributes and Operations:** For each class, define its attributes (properties) and operations (methods). Specify their data types and relationships with other classes.
  3. **Establish Relationships:** Determine the relationships between classes, such as associations, aggregations, compositions, and inheritance. Define multiplicity if applicable.
  4. **Draw the Diagram:** Use a diagramming tool or software to draw the Class Diagram, placing classes, attributes, operations, and relationships in their appropriate positions.

- **Problem Statement**

  To create a live cricket scoring app. Here, various types of functions are provided such as ball-by-ball scoring, professional scorecard etc.

  (Diagram on next page)


Team
+ teamId : String
+ teamName : String
+ teamCode : String
+ captain : String
+ viceCaptain : String
+ coach : String
+ manager : String
+ getTeamDetails() : String
Point
+ teamId : String
+ play : int
+ won : int
+ lost : int
+ tieOrDraw : int
/ point : int
+ getPoint() : int
+ getPointDetails() : String
Player
+ playerId : String
+ firstName : String
+ lastName : String
+ dateOfBirth: Date
/ age: int
+ isBat : boolean
+ isBowl : boolean
+ isWicketKeeper : boolean
+ teamId : String
+ getAge(dateOfBirth:Date,curDate:Date) : int
+ getPlayerDetails() : String
Match
+ matchId : String
+ matchNo : int
+ team1 : String
+ team2 : String
+ date : String
+ over : double
+ matchType : String
+ getMatchDetails() : String
Toss
+ matchId : String
+ tossWin : String
+ choose : String
+ getTossDetails() : String
Batting
+ playerId : String
+ position : int
- totalWicket : int
+ bowl : int
+ dot : int
+ one : int
+ two : int
+ three : int
+ four : int
+ six : int
/ run : int
+ isOut : boolean
+ outType : String
+ getRun() : int
+ getBattingDetails() : String
+ getTotalWicket() : int
Bowling
+ playerId : String
+ bowl : int
+ maiden : int
+ dot : int
+ one : int
+ two : int
+ three : int
+ four : int
+ six : int
+ wide : int
+ noBowl : int
+ legBy : int
/ run : int
/ over : double
/ economy : double
+ wicket : int
+ getRun() : int
+ getOver() : double
+ getEconomy() : double
+ getBowlingDetails() : String
Result
+ matchId : String
+ winTeam : String
+ looseTeam : String
+ runOrWicket : boolean
+ winValue : int
/ result : String
+ getResult(runOrWicket : boolean,winValue : int) : String
+ getResultDetails() : String

- **Class Diagram Best Practices**
  To ensure the effectiveness of Class Diagrams, consider these best practices:
  1. **Keep It Simple:** Avoid overcomplicating the diagram. Focus on the most critical classes, attributes, and relationships.
  2. **Consistency in Notations:** Maintain consistency in notations and naming conventions throughout the diagram. Use a standard naming convention for classes, attributes, and operations.
  3. **Documenting Details:** Include concise descriptions or notes in the diagram to provide additional context and explanations where necessary.

- **Real-World Applications of Class Diagrams**
  Class Diagrams find applications in various domains beyond software development:
  1. **Software Design and Development:** In software engineering, Class Diagrams are used to design and document the structure of software systems, aiding in code generation and system understanding.
  2. **Object Oriented Analysis and Design:** Class Diagrams are an essential tool in object-oriented analysis and design, helping model and define the architecture of systems.
  3. **Database Design:** In database design, Class Diagrams can be adapted to create Entity Relationship Diagrams (ERDs), which represent the structure of a database system.

<u>Conclusion:</u>

In conclusion, UML Class Diagrams play a critical role in visualizing and designing the static structure of a system. They provide a clear representation of classes, their attributes, operations, and relationships, making them a valuable asset in system design and development. Whether used in software engineering, object-oriented analysis and design, or database design, Class Diagrams serve as a foundation for creating well-structured and organized systems. Understanding and effectively employing this UML diagram can greatly enhance the quality and efficiency of system design and development projects.