UML Cheat Sheet

A concise cheat sheet covering the essential aspects of Unified Modeling Language (UML), including diagrams, relationships, and notation.

UML Diagrams Overview

Structural Diagrams

Class Diagram: Represents the static structure of a system, showing classes, attributes, operations, and relationships.
Object Diagram: Shows instances of classes and their relationships at a specific point in time.
Component Diagram: Illustrates the organization and relationships of software components.
Deployment Diagram: Depicts the physical deployment of software components to hardware nodes.
Package Diagram: Organizes model elements into packages to manage complexity.
Profile Diagram: Allows defining custom stereotypes, tagged values, and constraints to extend UML for specific domains.

Behavioral Diagrams

Use Case Diagram: Captures the functional requirements of a system from the user's perspective.
Activity Diagram: Models the flow of activities within a system or business process.
State Machine Diagram: Describes the states an object can be in and the transitions between those states in response to events.
Sequence Diagram: Illustrates interactions between objects in a time-ordered sequence.
Communication Diagram: Similar to sequence diagrams, but focuses on object relationships rather than time sequence. Also known as collaboration diagram.
Interaction Overview Diagram: Provides a high-level view of the interactions within a system, combining aspects of activity and sequence diagrams.
Timing Diagram: Shows the change in state or value of one or more objects over time.

Class Diagram Elements

Classes

Notation:	Represented as a rectangle divided into three sections: class name, attributes, and operations.
Attributes:	Characteristics or properties of a class. Indicated by name, type, and visibility (e.g., `+name: String`).
Operations:	Actions or functions that a class can perform. Indicated by name, parameters, and return type (e.g., `+getName(): String`).
Visibility:	`+` Public, `-` Private, `#` Protected, `~` Package.

Relationships

Association:	A general relationship between classes, indicated by a solid line. Can be unidirectional or bidirectional.
Aggregation:	A 'has-a' relationship representing a whole-part hierarchy, where the part can exist independently of the whole. Represented by a line with an open diamond at the whole end.
Composition:	A strong 'has-a' relationship where the part cannot exist independently of the whole. Represented by a line with a filled diamond at the whole end.
Generalization (Inheritance):	An 'is-a' relationship where one class inherits from another. Represented by a line with an open triangle at the parent class end.
Realization:	A relationship between an interface and a class that implements it. Represented by a dashed line with an open triangle at the interface end.
Dependency:	A weaker form of relationship indicating that one class uses or depends on another. Represented by a dashed line.

Use Case Diagram Elements

Actors

Represented as stick figures. Actors interact with the system but are external to it. Can be human users, external systems, or hardware devices.

Use Cases

Represented as ovals. Use cases are high-level descriptions of what a system should do from the actor's perspective.

Relationships

Association:	Represents interaction between an actor and a use case. A solid line connects the actor to the use case.
Include:	Indicates that one use case includes the functionality of another. Represented by a dashed line with an arrow pointing to the included use case and labeled `<<include>>`.
Extend:	Indicates that one use case extends the functionality of another. Represented by a dashed line with an arrow pointing to the extended use case and labeled `<<extend>>`.
Generalization:	An 'is-a' relationship between use cases, indicating that one use case inherits the behavior of another. Represented by a solid line with an open triangle pointing to the parent use case.

System Boundary

A rectangle that encloses the use cases, representing the boundary of the system.

Activity & Sequence Diagrams

Activity Diagram Elements

Initial Node:	Represents the starting point of the activity. Shown as a filled circle.
Activity:	Represents a task or action performed. Shown as a rounded rectangle.
Decision Node:	Represents a branching point in the activity flow. Shown as a diamond.
Merge Node:	Represents a point where multiple flows converge into one. Shown as a diamond.
Fork Node:	Splits a single flow of control into multiple concurrent flows. Shown as a bar.
Join Node:	Synchronizes multiple concurrent flows into a single flow. Shown as a bar.
Final Node:	Represents the end of the activity. Shown as a bullseye.

Sequence Diagram Elements

Lifeline:	Represents the existence of an object over time. Shown as a vertical dashed line.
Activation Box:	Indicates when an object is performing an action. Shown as a thin rectangle on the lifeline.
Message:	Represents communication between objects. Shown as an arrow from one lifeline to another.
Synchronous Message:	The sender waits for a response. Shown as a solid arrow.
Asynchronous Message:	The sender does not wait for a response. Shown as an open arrow.
Return Message:	Represents the response to a synchronous message. Shown as a dashed arrow.