Experiment 11

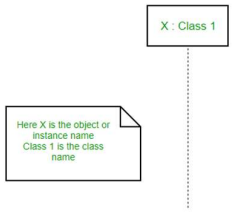
AIM: Develop Detailed Sequence Diagrams / Communication diagrams for each use case showing interactions among all the three-layer objects

DESCRIPTION:

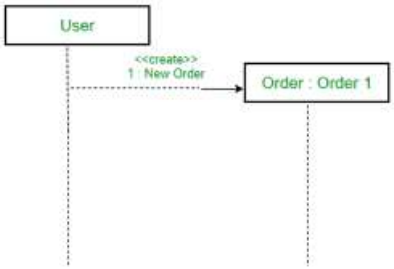
A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems

Lifelines

A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. The standard in UML for naming a lifeline follows the following format – Instance Name : Class



Messages – Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram



Create message :

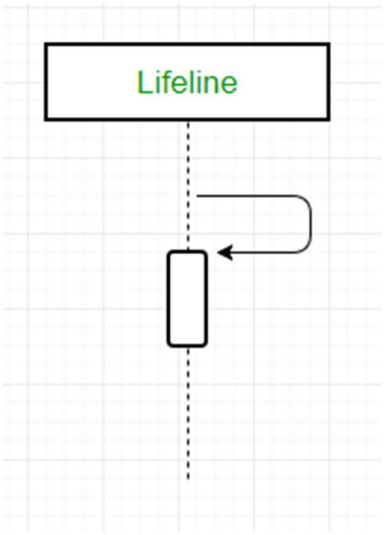
We use a Create message to instantiate a new object in the sequence diagram. There are situations when a particular message call requires the creation of an object. It is represented with a dotted arrow and create word labelled on it to specify that it is the create Message symbol. For example – The creation of a new order on a e-commerce website would require a new object of Order class to be created

Delete Message

We use a Delete Message to delete an object. When an object is deallocated memory or is destroyed within the system we use the Delete Message symbol. It destroys the occurrence of the object in the system.It is represented by an arrow terminating with a x. For example – In the scenario below when the order is received by the user, the object of order class can be destroyed.

Self Message

Certain scenarios might arise where the object needs to send a message to itself. Such messages are called Self Messages and are represented with a U shaped arrow.



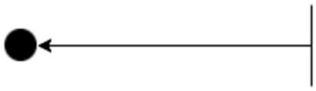
Found Message

A Found message is used to represent a scenario where an unknown source sends the message. It is represented using an arrow directed towards a lifeline from an end point. For example: Consider the scenario of a hardware

Reply Message

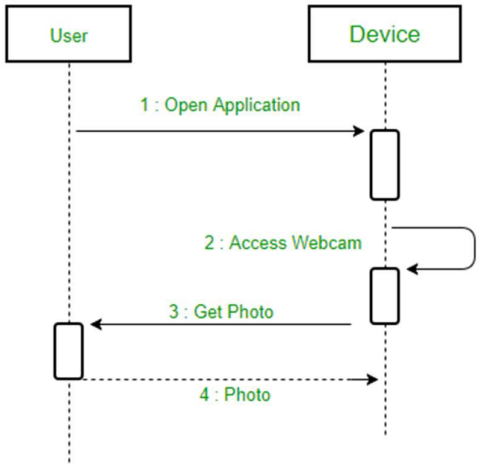
Reply messages are used to show the message being sent from the receiver to the sender. We represent a return/reply message using an open arrowhead with a dotted line. The interaction moves forward only when a reply message is sent by the receiver.

Lost Message – A Lost message is used to represent a scenario where the recipient is not known to the system. It is represented using an arrow directed towards an end point from a lifeline. For example: Consider a scenario where a warning is generated.



Synchronous messages

A synchronous message waits for a reply before the interaction can move forward. The sender waits until the receiver has completed the processing of the message. The caller continues only when it knows that the receiver has processed the previous message i.e. it receives a reply message. A large number of calls in object oriented programming are synchronous. We use a solid arrow head to represent a synchronous message.



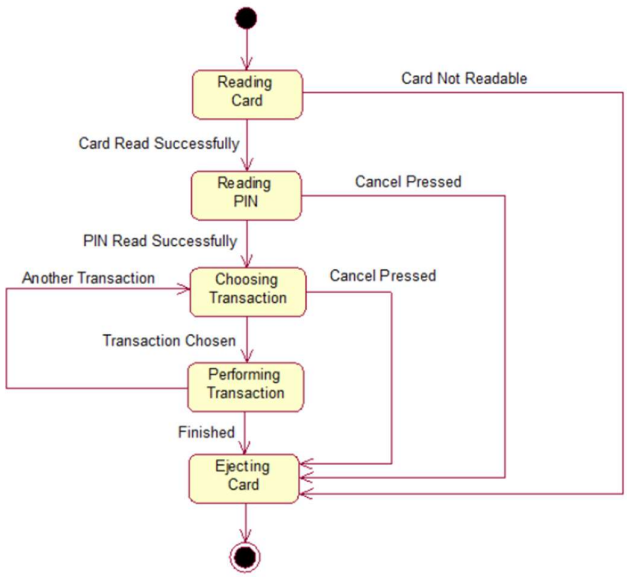
Experiment 12

AIM: Develop sample diagrams for other UML diagrams - state chart diagrams, activity diagrams, component diagrams and deployment diagrams

DESCRIPTION:

Statechart diagram

- A statechart diagram shows a state machine, consisting of states, transitions, events, and activities
- Statechart diagrams address the dynamic view of a system
- They are especially important in modeling the behavior of an interface, class, or collaboration and emphasize the event-ordered behavior of an object

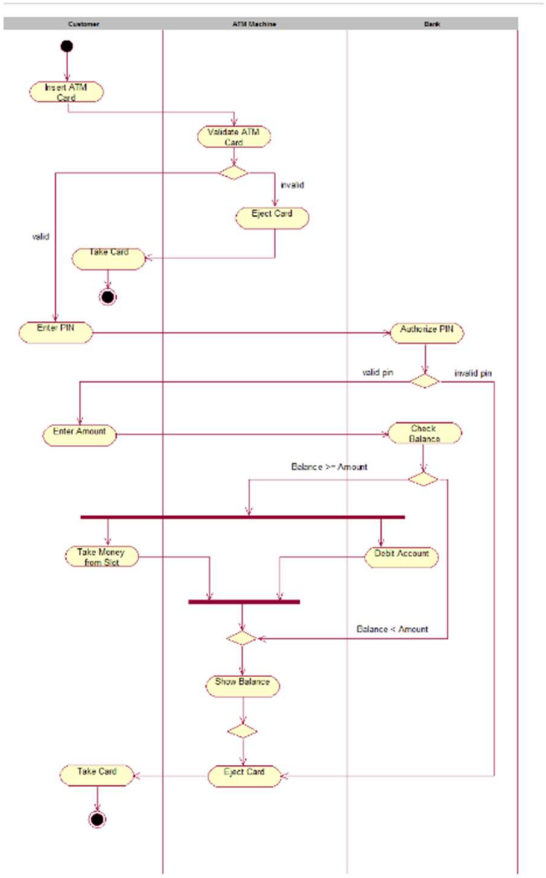


Activity diagram

An activity diagram is a special kind of a statechart diagram that shows the flow from activity to activity within a system

Activity diagrams address the dynamic view of a system

They are especially important in modeling the function of a system and emphasize the flow of control among objects



Component diagram

- A component diagram shows the organizations and dependencies among a set of components.
- Component diagrams address the static implementation view of a system
- They are related to class diagrams in that a component typically maps to one or more classes, interfaces, or collaborations

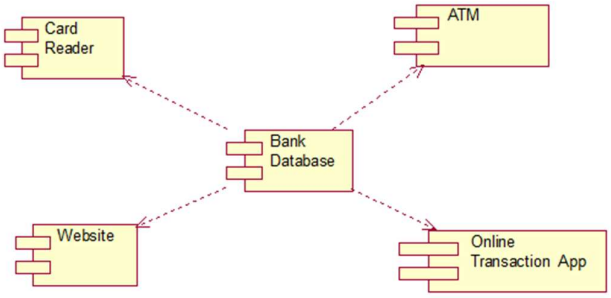


Fig. Component Diagram for ATM

Deployment diagram

- A deployment diagram shows the configuration of run-time processing nodes and the components that live on them

Deployment diagrams address the static deployment view of an architecture

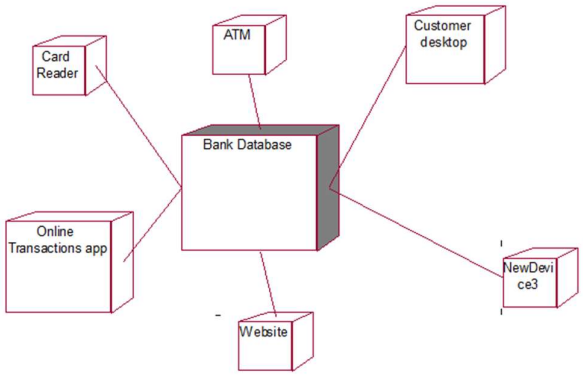


Fig. deployment diagram for ATM

Experiment 13

Aim: Implement decision trees using 'iris' dataset using package party and 'rpart'

Description:

The Iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris Setosa, Versicolor, and Virginica. The data set has 150 cases (rows) and 5 variables (columns) named Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species.

Program:

```
library(dplyr)

library(tidyr)

library(ggplot2)

library(rpart)

library(rpart.plot)

library(caret)

iris_tidy <- read.csv(file="../input/iris-flower-data-set-cleaned/iris_tidy.csv")

iris_df <- data.frame(iris_tidy)

ggplot(iris_df, aes(x = Measure, y = Value, col = Part)) +

  geom_jitter() +

  facet_grid(. ~ Species)
```

Experiment 14

Aim: Develop Use case Packages and identify relationships between use cases and represent them.Refine domain class model by showing all the associations among classes.

Description:

Domain Models

A domain model is a *visual* representation of conceptual classes or real-world objects in a domain of interest

Domain models are also called conceptual models, domain object models, and analysis object models.

UP defines a Domain Model as one of the artifacts that may be created in the Business Modeling discipline.

Using UML notation, a domain model is illustrated with a set of class diagrams in which no operations are defined.

It may show:

domain objects or conceptual classes

associations between conceptual classes

e.g. a partial domain model

- conceptual class of *Payment* and *Sale* are significant in this domain,
- Payment is related to a Sale in a way that is meaningful to note
  - Sale has a date and time.

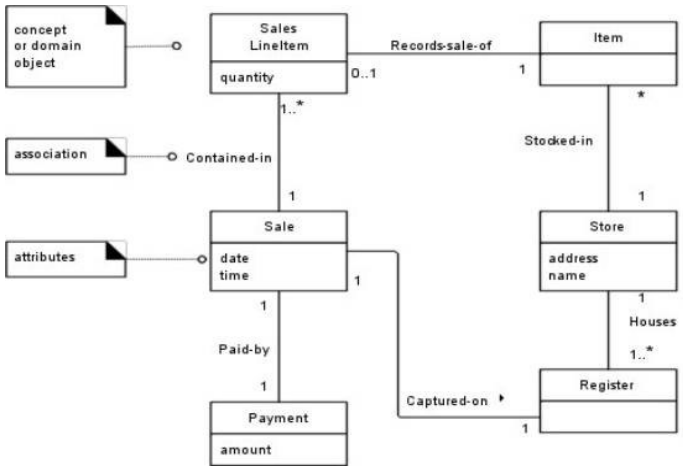
Key Idea: Domain Model—A Visual Dictionary of Abstractions

It visualizes and relates some words or conceptual classes in the domain.

It depicts an abstraction of the conceptual classes

The model displays a partial view, or abstraction, and ignores uninteresting (to the modelers) details.

May be considered a visual dictionary of the noteworthy abstractions, domain vocabulary, and information content of the domain.

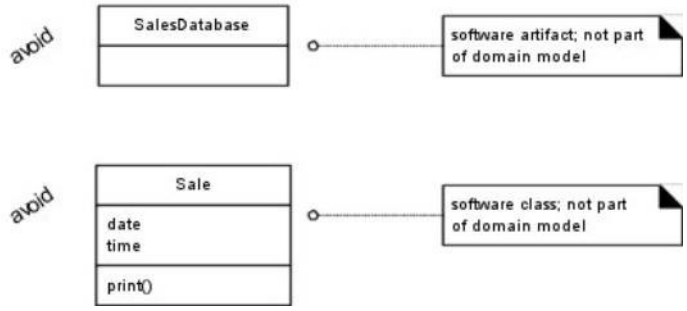


Domain Models Are not Models of Software Components

A domain model is a visualization of things in the real world domain of interest

Following elements not suitable in a domain model:

Software artifacts, such as a window or a database, unless the domain being modeled is of software concepts, such as a model of graphical user interfaces.



Use cases represent the externally visible behaviors, or functional aspects, of the system. They consist of the abstract, uninterpreted interactions of the system with external entities. This means that the content of use cases is not used for code generation.

- Creating use cases
  A use case is displayed in a use case diagram as an oval containing a name.
- Modifying the features of a use case
- Adding attributes to a use case
  Because a use case is a stereotyped class, it can have attributes. No code is generated for these attributes.
- Adding operations to a use case
  Because a use case is a stereotyped class, it can have operations.