RECORD NOTE BOOK

NAME :

BRANCH :

YEAR :

ROLL NUMBER :

CASE TOOLS LABORATORY
B.TECH II YEAR, I-SEMESTER
INSTRUCTOR : Dr . TIRIMULA RAO BENALA

CERTIFICATE

Certified that this is a bonafide record of work done by Mr./Kumari …….

of …… Class in ……. Laboratory of Jawaharlal Nehru

Technological University Gurajada Vizianagaram,During the Year …….

No of experiments done and certified :

Lecturer in-charge

Date :

HEAD OF THE DEPARTMENT

Table of Contents :

AIM: SOFTWARE REQUIREMENT SPECIFICATIONS FOR E-TICKETING

1. Introduction

1. Introduction

An electronic ticket, more efficient method of ticket entry, processing and marketing for companies in the airline, railways and other transport and entertainment industries. Many sports, concert venues, and cinemas use electronic ticketing for their events. Elec- tronic tickets, or ”eTickets” as they are sometimes referred, are often delivered as PDFs or another downloadable format that can be received via email or through a mobile app. Electronic tickets allow spectators to download their tickets, as opposed to waiting for physical tickets to arrive in the mail. A printed copy of these tickets or a digital copy on a mobile phone should be presented on coming to the venue.

1.1 Purpose

This document is intended for the following of people:

- Developers for the purpose of the maintenance and update of the website
- Documentation writers.
- Movie theatre management.
- Testers.

1.2 Document Conventions

This document uses Calibri font:

- Main sections: Bold, font size 14.
- Subsections: Bold, font size 14.
- Acronyms and abbreviations are in Appendix A.

1.3 Intended Audience and Reading Suggestions

This document is intended to be read by the customer. This is A technical document and the terms should be understood by the customer. The designer needs to understand this document fully so that they can draft A design document using this SRS presented to them by the analyst.

1.4 Product Scope

The Online Ticket Reservation System will be used by passengers who may be familiar or not familiar to the online train reservation process thus the scope of the project must be user friendly for both passengers and admin.

1.5 References

- www.google.co.in
- www.wikipedia.com
- www.academia.edu

2. Overall Description

2.1 Product Perspective

This software allows the user to access different theatres and shows in them through an application. This software also allows to search user's liked movie and displays the available shows in different theatres. The user is supposed to select wished seats and book them by making an online transaction.

2.2 Product Functions

- **Administrators:** Can add or delete shows and reject users who do not accept terms and conditions.
- **Users:** Can view, book, and cancel seats and pay through online transactions.

2.3 Operating Environment

ticketing: A ticket is produced with seat numbers, time and place of show on it and also a QR code. Cancellation: User also may un book the seats and get a 50

2.4 Design and Implementation Constraints

The system is internet based system, a webapp that should be developed to support any web browser to be used. Passenger Account and Payment for train ticket should be secure, the system design should include a lot of DB and SYSTEM validation. Developers needs to be always there to support the delivered system in terms of validation and maintenance.

2.5 Assumptions and Dependencies

The display screen shall be of 10" VGA color type. The display screen shall have 250 color resolution. The display screen shall also support touch screen facility. The speakers shall support Yamaha codecs. The keypad shall consist of 16 tactile keys. There shall be 8 tactile function keys. The keyboard will be weather resistant. The transaction receipt shall be 3.1"x 6". The statement receipt shall be 4.2" x 12". The deposit envelopes shall be 9" long and 4" wide

3. External Interface Requirements

3.1 User Interface

The system should be user-friendly and accessible via major web browsers.

3.2 Hardware Interface Requirements

The ATM power supply shall have a 10/220 V AC manual switch. The ATM card should have the following physical dimensions:- The card reader shall be a magnetic stripe reader. The slot for a card in the card reader may include an extra indentation for the embossed area of the card. There shall be a 40 column dot matrix receipt printer. The statement dispenser shall be a maximum of 5" width and 0.5" thickness. The envelope depository

shall be a maximum of 4.5” width, 10” length and 0.5” thickness. Screen resolution of at least 800X600-required for proper and complete viewing of screens.

3.3 Software Interface Requirements

In order to perform various different functions, this software needs to interact with var- ious other softwares. So there are certain software interface requirements that need to be fulfilled which are listed as follows: The transaction management software used to manage the transaction and keep track of resources shall be BMS version 2.0. The card management software used to verify pin no and login shall be CMS version 3.0

3.4 Communication Interface Requirements

The machine needs to communicate with the main branch for each session for various functions such as login verification, account access etc. so the following are the various communication interface requirements that are needed to be fulfilled in order to run the software successfully

4. System Features

The Online Ticket Reservation System comprises of two main features, namely, internet connectivity which will enables users to communicate with the server through a browser or web agent, And secondly the system requires database service to store the user’s data. In a nutshell this system is web application and thus is only operational in an internet enabled environment.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The system is required a fair amount of speed especially while browsing through the trains list and availability of seats. The database shall be able to accommodate a minimum of 100000 records of passengers. The software will support multiple users.

5.2 Safety Requirements

- 1 data protection
- 2 secure transaction
- 3 user authentication and authorization
- 4 system integrity and availability
- 5 user safety and experience
- 6 emergency response and incident management
- 7 secure software development life cycle

5.3 Security Requirements

The main security concern is for users accounts and payment details, hence proper login Mechanism should be used to avoid hacking. The online ticket reservation system shall not disclose personal information of passengers to unauthorized users or the public.

5.4 Software Quality Attributes

Availability: The database will have to be available to passengers 24/7.

5.5 Business Rules

The online ticket reservation system shall include two types of accounts: the administra- tors and the passengers. To log in to the system user name and password is required.

Use Case	Description	Actors	Preconditions	Postconditions
User Registration	Allows a new user to register an account on the platform.	User	User must not already have an account.	User account is created.
Login	Allows users to log in to their accounts.	User	User must have an existing account.	User is authenticated and logged in.
Search Events	Allows users to search for available events by location, date, or category.	User	User must be logged in.	Event search results are displayed.
View Event Details	Shows detailed information about a selected event.	User	Event must be available in the system.	Event details are displayed.
Book Ticket	Allows users to book a ticket for a selected event.	User	User must be logged in, and tickets must be available.	Ticket booking is initiated.
Select Seat	Allows users to choose specific seats for an event (if applicable).	User	User has selected an event and initiated booking.	Seats are selected and locked.
Make Payment	Allows users to pay for the selected tickets.	User, Payment Gateway	User has selected tickets and provided payment info.	Payment is processed, and booking is confirmed.
Receive E-Ticket	Provides the user with an e-ticket upon successful payment.	User	Payment must be successful.	E-ticket is generated and sent to user.
Cancel Booking	Allows users to cancel an existing booking.	User	User must have an existing, valid booking.	Booking is canceled, and refund is initiated if eligible.
View Booking HisTory	Allows users to view their past and current bookings.	User	User must be logged in.	Booking history is displayed.
Send Notifications	Sends notifications to users about booking status, reminders, or updates.	System, User	User must have opted for notifications.	Notifications are sent to the user.
Generate Reports	Generates reports on bookings, cancellations, and event statistics.	Admin	Admin must be logged in.	Reports are generated.
Manage Events	Allows admins to create, update, or delete events on the platform.	Admin	Admin must be logged in.	Events are managed in the system.
Manage Users	Allows admins to view, update, or block users if necessary.	Admin	Admin must be logged in.	User information is managed.

Aim : Implement Use Case Diagram using Planttext UML Editor

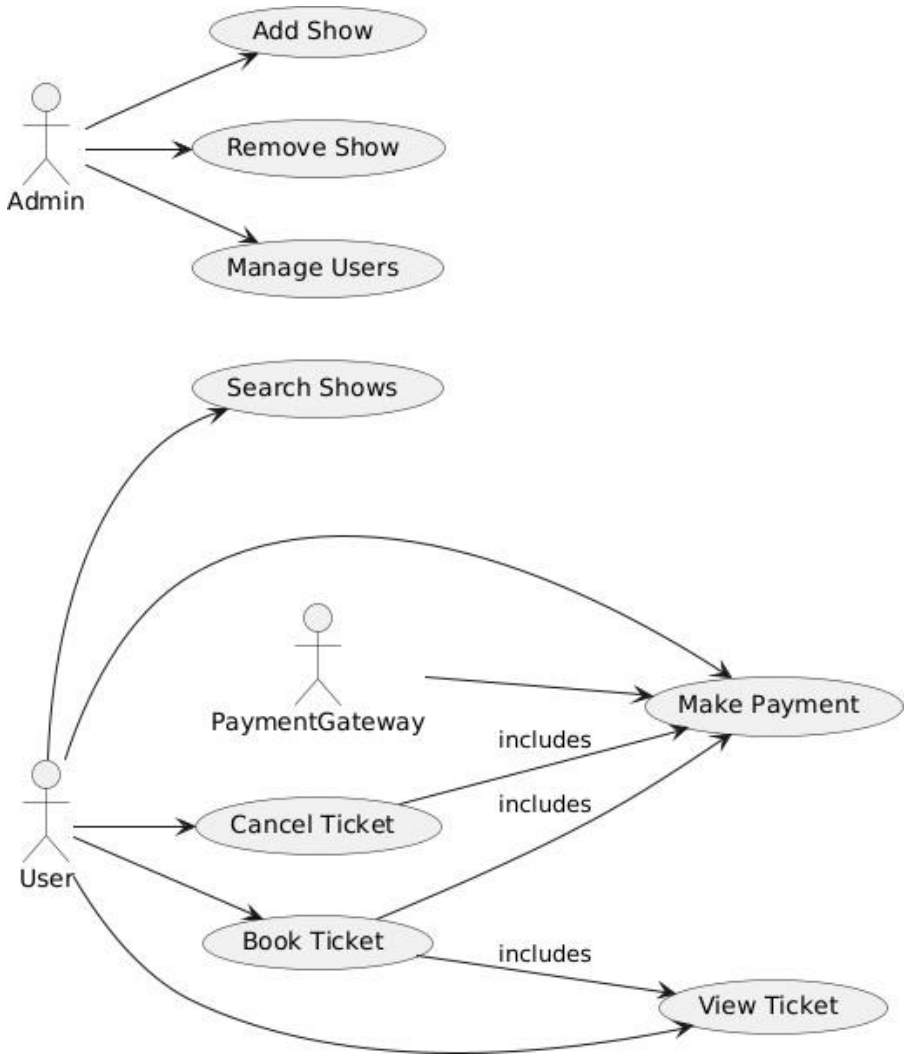


Figure 1: USECASE DIAGRAM FOR E-TICKETING

File Manager **Default Diagram** Samples Refresh

```
@startuml

left to right direction

' Define actors
actor User
actor Admin
actor PaymentGateway

' Define use cases
usecase "Search Shows" as UC_SearchShows
usecase "Book Ticket" as UC_BookTicket
usecase "Cancel Ticket" as UC_CancelTicket
usecase "Make Payment" as UC_MakePayment
usecase "View Ticket" as UC_ViewTicket
usecase "Add Show" as UC_AddShow
usecase "Remove Show" as UC_RemoveShow
usecase "Manage Users" as UC_ManageUsers

' Define relationships between actors and use cases
User --> UC_SearchShows
User --> UC_BookTicket
User --> UC_CancelTicket
User --> UC_ViewTicket
User --> UC_MakePayment

Admin --> UC_AddShow
Admin --> UC_RemoveShow
Admin --> UC_ManageUsers

UC_BookTicket --> UC_MakePayment : includes
UC_CancelTicket --> UC_MakePayment : includes
UC_BookTicket --> UC_ViewTicket : includes

PaymentGateway --> UC_MakePayment

@enduml
```

Figure 2: USECASE DIAGRAM CODE FOR E-TICKETING

Aim : Implement Class Diagram using Plant text UML Editor

Class Diagram for E-Ticketing System

The e-ticketing system consists of several key entities, their attributes, and relationships, described below:

1. User

Description: Represents an individual who interacts with the e-ticketing system. Users can be customers or admins.

Attributes:

- userID (PK): Unique identifier for the user.
- name: Name of the user.
- email: Email address of the user.
- password: Password for user authentication.
- phoneNumber: Contact phone number of the user.
- userType: Type of user (Admin, Customer, etc.).

Relationships:

- **1-to-Many** with Booking: A user can make multiple bookings.
- **1-to-Many** with Payment: A user can make multiple payments for bookings.

2. Event

Description: Represents an event for which tickets are available.

Attributes:

- eventID (PK): Unique identifier for the event.
- name: Name of the event.
- date: Date of the event.
- time: Time of the event.
- location: Location where the event is held.
- description: Detailed description of the event.
- availableTickets: Number of tickets available for the event.
- price: Price of a ticket for the event.

Relationships:

- **1-to-Many** with Ticket: An event can have multiple tickets available.
- **1-to-Many** with Booking: Multiple bookings can be made for the same event.

3. Ticket

Description: Represents a specific ticket for an event that can be booked.

Attributes:

- ticketID (PK): Unique identifier for the ticket.
- eventID (FK): Reference to the event that this ticket belongs to.
- seatNumber: Seat number associated with the ticket.
- ticketType: Type of the ticket (Regular, VIP, etc.).
- price: Price of the ticket.
- status: Current status of the ticket (Available, Booked, etc.).

Relationships:

- **1-to-1** with Booking: Each ticket is associated with a specific booking.
- **Many-to-1** with Event: Multiple tickets are associated with one event.

4. Booking

Description: Represents a user's reservation for one or more tickets for an event.

Attributes:

- bookingID (PK): Unique identifier for the booking.
- userID (FK): Reference to the user making the booking.
- eventID (FK): Reference to the event being booked.
- bookingDate: Date when the booking was made.
- status: Status of the booking (Confirmed, Canceled, etc.).

Relationships:

- **Many-to-1** with User: A user can make multiple bookings.
- **Many-to-1** with Event: A booking is for a specific event.
- **1-to-1** with Ticket: A booking is associated with one ticket.

5. Payment

Description: Represents the payment made by the user for the booking.

Attributes:

- paymentID (PK): Unique identifier for the payment.
- bookingID (FK): Reference to the booking being paid for.
- userID (FK): Reference to the user making the payment.
- amount: Total amount paid.
- paymentDate: Date when the payment was made.
- paymentMethod: Method of payment (Credit Card, PayPal, etc.).
- status: Status of the payment (Completed, Pending, etc.).

Relationships:

- **Many-to-1** with User: A user can make multiple payments.
- **1-to-1** with Booking: A payment is linked to one specific booking.

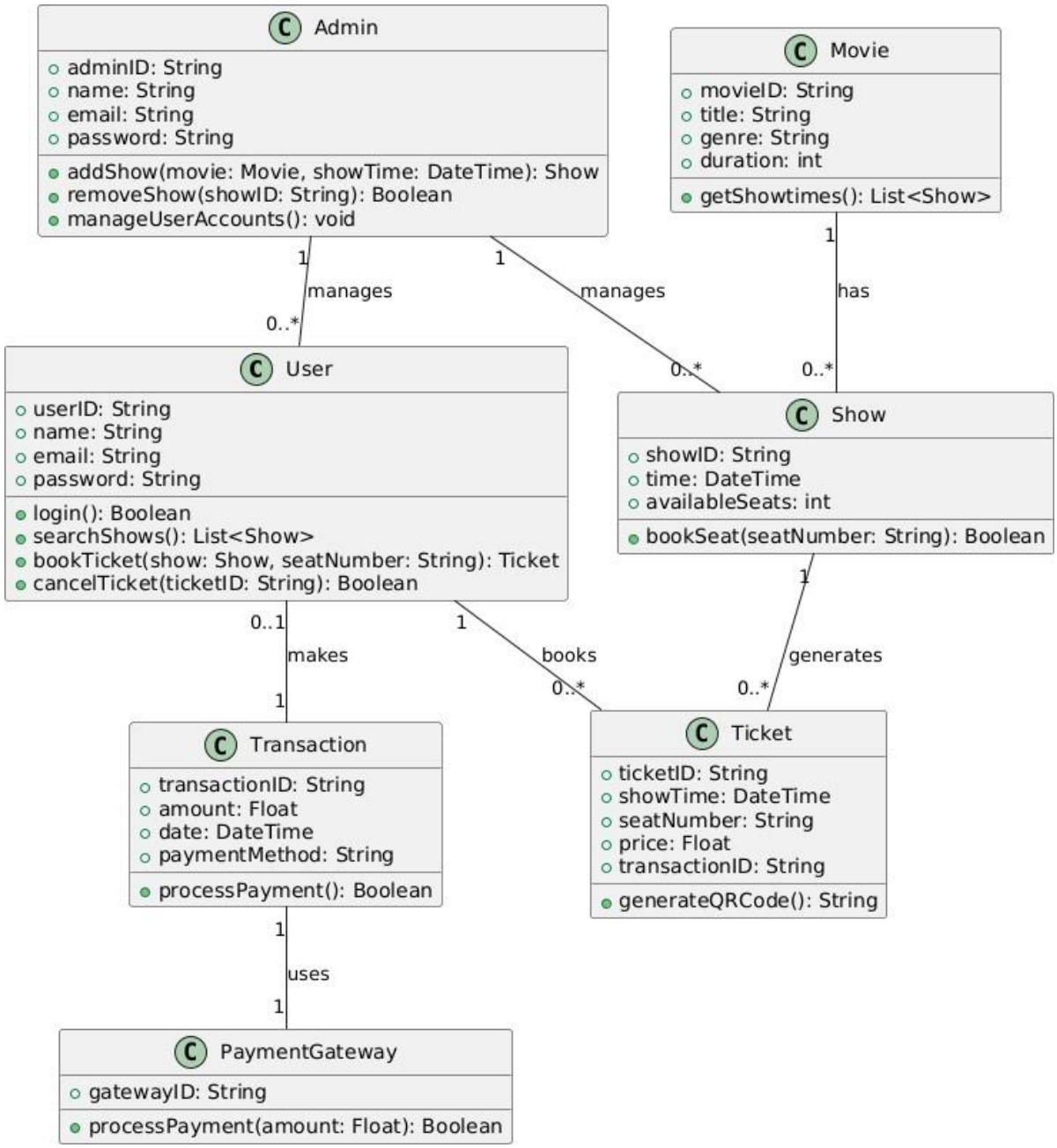


Figure 3: CLASS DIAGRAM FOR E-TICKETING

```
Manager  Default Diagram                                   Sample
Use the File Manager to save and work with files stored in the local browser cache.

' Define classes
class User {
    +userID: String
    +name: String
    +email: String
    +password: String
    +login(): Boolean
    +searchShows(): List<Show>
    +bookTicket(show: Show, seatNumber: String): Ticket
    +cancelTicket(ticketID: String): Boolean
}

class Admin {
    +adminID: String
    +name: String
    +email: String
    +password: String
    +addShow(movie: Movie, showTime: DateTime): Show
    +removeShow(showID: String): Boolean
    +manageUserAccounts(): void
}

class Ticket {
    +ticketID: String
    +showTime: DateTime
    +seatNumber: String
    +price: Float
    +transactionID: String
    +generateQRCode(): String
}

class Movie {
    +movieID: String
    +title: String
```

```
File Manager  Default Diagram
    +price: Float
    +transactionID: String
    +generateQRCode(): String
}

class Movie {
    +movieID: String
    +title: String
    +genre: String
    +duration: int
    +getShowtimes(): List<Show>
}

class Show {
    +showID: String
    +time: DateTime
    +availableSeats: int
    +bookSeat(seatNumber: String): Boolean
}

class Transaction {
    +transactionID: String
    +amount: Float
    +date: DateTime
    +paymentMethod: String
    +processPayment(): Boolean
}

class PaymentGateway {
    +gatewayID: String
    +processPayment(amount: Float): Boolean
}

' Define relationships
User "1" -- "0..*" Ticket : books
User "0..1" -- "1" Transaction : makes
Admin "1" -- "0..*" Show : manages
Admin "1" -- "0..*" User : manages
Transaction "1" -- "1" PaymentGateway : uses
Movie "1" -- "0..*" Show : has
Show "1" -- "0..*" Ticket : generates

@enduml
```

Figure 4: CLASS DIAGRAM CODE FOR E-TICKETI

Aim : Implement Sequence Diagram using Plant text UML

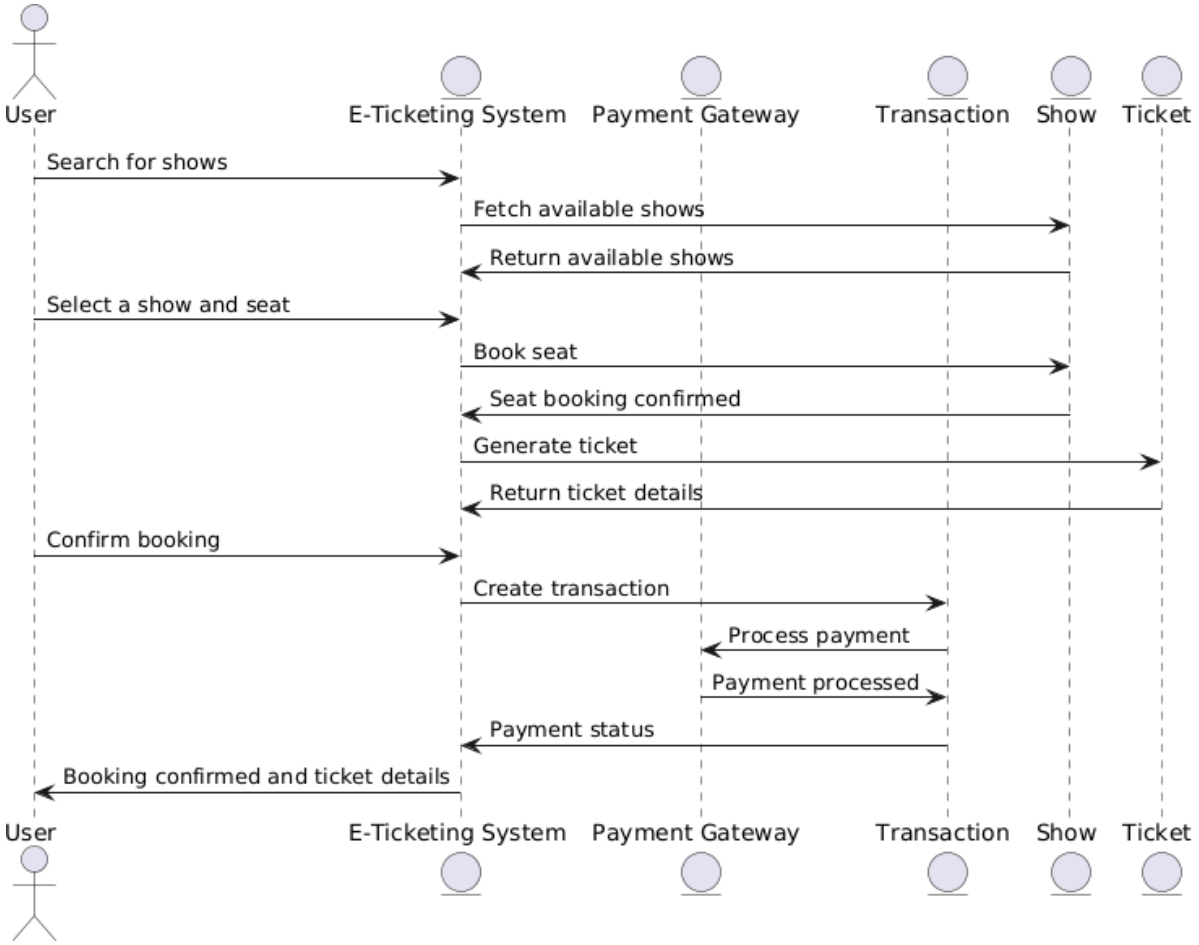


Figure 5: SEQUENCE DIAGRAM FOR E-TICKETING

```
@startuml

actor User
entity "E-Ticketing System" as System
entity "Payment Gateway" as PaymentGateway
entity "Transaction" as Transaction
entity "Show" as Show
entity "Ticket" as Ticket

User -> System : Search for shows
System -> Show : Fetch available shows
Show -> System : Return available shows
User -> System : Select a show and seat
System -> Show : Book seat
Show -> System : Seat booking confirmed
System -> Ticket : Generate ticket
Ticket -> System : Return ticket details
User -> System : Confirm booking
System -> Transaction : Create transaction
Transaction -> PaymentGateway : Process payment
PaymentGateway -> Transaction : Payment processed
Transaction -> System : Payment status
System -> User : Booking confirmed and ticket details

@enduml
```

Figure 6: SEQUENCE DIAGRAM CODE FOR E- TICKETING

Aim : Implement Sequence Diagram using Plant text UML Editor

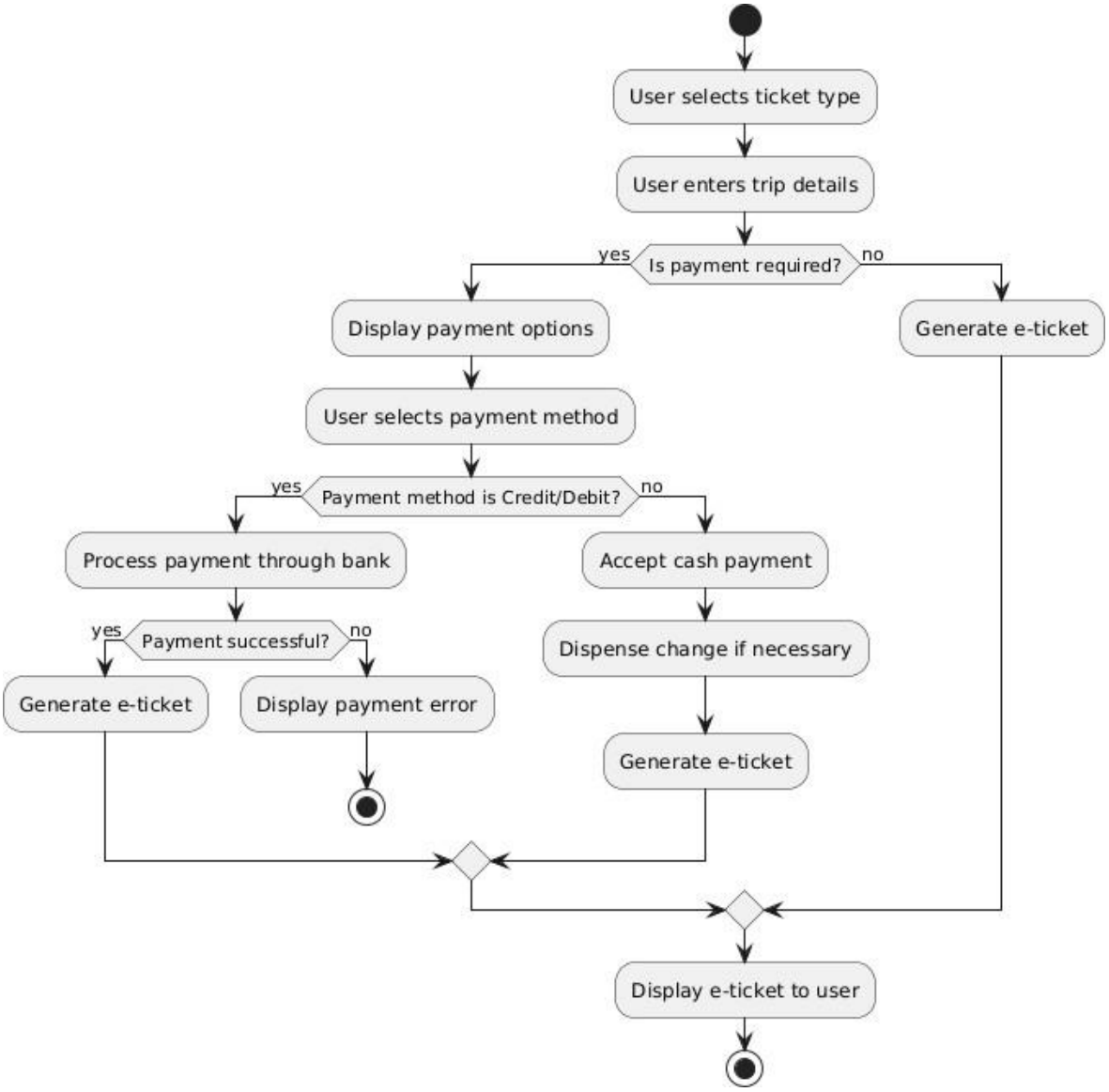


Figure 7: ACTIVITY DIAGRAM FOR E-TICKETING

Figure 8: ACTIVITY DIAGRAM CODE FOR E-TICKETING

```
@startuml
start

:User selects ticket type;
:User enters trip details;

if (Is payment required?) then (yes)
    :Display payment options;
    :User selects payment method;

    if (Payment method is Credit/Debit?) then (yes)
        :Process payment through bank;
        if (Payment successful?) then (yes)
            :Generate e-ticket;
        else (no)
            :Display payment error;
            stop
        endif
    else (no)
        :Accept cash payment;
        :Dispense change if necessary;
        :Generate e-ticket;
    endif
else (no)
    :Generate e-ticket;
endif

:Display e-ticket to user;

stop
@enduml
```

Aim : Implement Collaboration Diagram using Plant text UML Editor

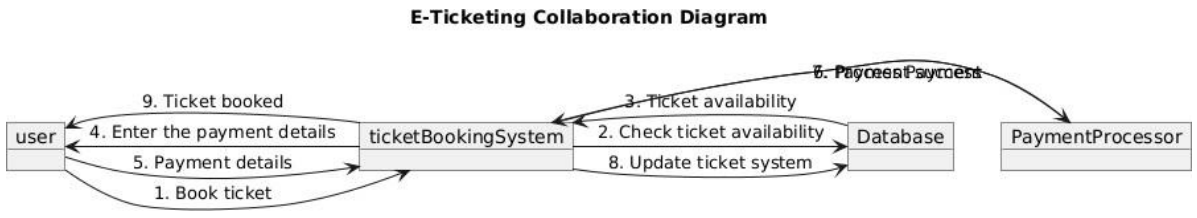


Figure 9: COLLOBRATION DIAGRAM FOR E-TICKETING
Figure 10: COLLOBRATION DIAGRAM CODE FOR E-TICKETING

Figure 10: COLLOBRATION DIAGRAM CODE FOR E-TICKETING

```
@startuml
title E-Ticketing Collaboration Diagram

object user
object ticketBookingSystem
object Database
object PaymentProcessor

user -> ticketBookingSystem: 1. Book ticket
ticketBookingSystem -> Database: 2. Check ticket availability
Database -> ticketBookingSystem: 3. Ticket availability
ticketBookingSystem -> user: 4. Enter the payment details
user -> ticketBookingSystem: 5. Payment details
ticketBookingSystem -> PaymentProcessor: 6. Process Payment
PaymentProcessor -> ticketBookingSystem: 7. Payment success
ticketBookingSystem -> Database: 8. Update ticket system
ticketBookingSystem -> user: 9. Ticket booked

@enduml
```

Aim : Illustrate Usage of Cost Constructive Model

6 What is Constructive Cost Model

The Constructive Cost Model (COCOMO) is a software cost estimation model developed by Barry Boehm in the early 1980s. It's widely used to predict the cost, effort, and schedule needed to develop software projects based on project size and other factors. The model helps project managers and software engineers estimate the resources required, based on various project characteristics and team dynamics.

6.1 Classification Basic

COCOMO:

This is the simplest version, focusing on estimating project effort and cost based solely on the size of the software (usually measured in thousands of lines of code, or KLOC). Basic COCOMO has three modes based on project complexity:

- Organic Mode: For small, straightforward projects with small teams and minimal technical constraints.
- Semi-Detached Mode: For medium-sized projects with mixed constraints.
- Embedded Mode: For complex projects with tight hardware, software, and operational constraints.

Intermediate COCOMO:

This version adds 15 additional cost drivers to account for different project attributes, such as product, hardware, personnel, and project characteristics. It allows for more accurate estimates by considering factors like team experience, software reliability, and system complexity. **Detailed COCOMO:**

This version breaks down the project into individual components or modules and applies intermediate COCOMO to each, considering specific phase-sensitive cost drivers. It provides even greater precision but requires more detailed information about the project structure

.

6.2 Problems Related COCOMO

6.2.1 Problem 1

Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of a software developer is Rs.15,000 per month. Determine the effort required to develop the software product , the nominal development time, and the cost to develop the product.

Solution :
Given that this is a package of a Organic type. we can calculate Effort Required and the Cost to Develop the product using the formula

$$\text{Effort (PM)} = a \times (\text{KLOC})^b$$

where:

- a and b are constants based on the type of project (e.g., Organic, Semi-Detached, or Embedded),
- KLOC is the estimated number of lines of code (in thousands),
- PM represents Person-Months of effort.

a = 2.4 b = 1.05s

$$\text{Development Cost} = \text{Effort (PM)} \times \text{Cost per Month}$$

Software Development (Elaboration and Construction) Staffing Profile

Effort = 132.9 Person-months
Schedule = 17.4 Months
Cost = $23644

Total Equivalent Size = 32000 SLOC
Effort Adjustment Factor (EAF) = 1.00

Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	8.0	2.2	3.7	$1419
Elaboration	31.9	6.5	4.9	$5675
Construction	101.0	10.9	9.3	$17970
Transition	15.9	2.2	7.3	$2837

Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	1.1	3.8	10.1	2.2
Environment/CM	0.8	2.6	5.1	0.8
Requirements	3.0	5.7	8.1	0.6
Design	1.5	11.5	16.2	0.6
Implementation	0.6	4.1	34.3	3.0
Assessment	0.6	3.2	24.2	3.8
Deployment	0.2	1.0	3.0	4.8

ss

6.2.2 Problem 2

Software package is required by a company to mine existing customer data to select prospective customer for a new launch - Estimate to be 30,000 lines of source code of effort - Assume competent developers can be hired at Rs.50,000/-per month - However, commercial offering supporting almost all of the required features cost Rs.100,000/**Solution**

Given that this is a package to mine customer data, it would likely fall under the SemiDetached mode a = 3.0 b = 1.12

Effort = 123.8 Person-months
Schedule = 17.0 Months
Cost = $73416

Total Equivalent Size = 30000 SLOC
Effort Adjustment Factor (EAF) = 1.00

Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	7.4	2.1	3.5	$4405
Elaboration	29.7	6.4	4.7	$17620
Construction	94.1	10.6	8.9	$55796
Transition	14.9	2.1	7.0	$8810

Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	1.0	3.6	9.4	2.1
Environment/CM	0.7	2.4	4.7	0.7
Requirements	2.8	5.3	7.5	0.6
Design	1.4	10.7	15.1	0.6
Implementation	0.6	3.9	32.0	2.8
Assessment	0.6	3.0	22.6	3.6
Deployment	0.2	0.9	2.8	4.5

As commercial is far more affordable it is recommended.

6.2.3 Problem 3

Consider a company is developing a software package for an educational institute that would automate various book keeping activities associated with the institute academic activities such as course registration and grading. The institute has already installed other software applications that automate it's various activity areas such as stores and purchase ,accounting and faculty payroll which are already in operation. The size of the code that is expected to be return for academic package is 10,000 SLOC. The package to be developed by a vendor is very similar to software developed by same vendor for different client. The software to be developed needs to be seamlessly work with other applications running at institute and has to used existing DBMS and other software components. -The requirements for academic package are clear and unlike to change. -The development team put together by the vendor is collocated and cohesive. -Other aspects of project such as required reliability and product complexity, required reusability, platform difficulty, personal capability, facilities available and scheduled pressure are nominal. -Determine the effort required by the vendor assuming that it uses ad hoc development practices. -Compare the effort to what would be incurred if the vendor has high process maturity. **Solution:**

this project could be classified as Semi-Detached
a = 3.0 b = 1.12

By calculate Effort (As the cost for hiring is not mentioned lets consider in personmonth(PM))

Effort = 37.0 Person-months
Schedule = 11.6 Months
Cost = $0

Total Equivalent Size = 10000 SLOC
Effort Adjustment Factor (EAF) = 1.00

Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	2.2	1.4	1.5	$0
Elaboration	8.9	4.3	2.0	$0
Construction	28.1	7.2	3.9	$0
Transition	4.4	1.4	3.1	$0

Software Effort Distribution for RUP/MBASE (Person-Months

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.3	1.1	2.8	0.6
Environment/CM	0.2	0.7	1.4	0.2
Requirements	0.8	1.6	2.2	0.2
Design	0.4	3.2	4.5	0.2
Implementation	0.2	1.2	9.6	0.8
Assessment	0.2	0.9	6.7	1.1
Deployment	0.1	0.3	0.8	1.3

If the vendor has high process maturity (assuming a 20 percentage reduction in effort), the effort would be approximately:

31.64 Person-Months (PM).

Aim : Illustrate Installation Procedure of Eclipse Oxygen

Introduction

These Eclipse installation instructions are used by students enrolled in the Master of Science in Software Development and other Computer Science Department programs in both on-campus and online programs...

8 Eclipse Overview

Eclipse is one of the most popular Java application development environments for desktop and Android applications...

9 Downloading and Installing Eclipse

9.1 Downloading Eclipse

Go to https://www.eclipse.org/downloads/packages/installer to get started downloading Eclipse.The Eclipse Foundation regularly updates their website, so what you see may be different than the following.

Next, browse to where you downloaded your executable in Windows Explore. Alternatively, your browser may give you an option to run it directly,without the needed to navigate in Windows Explorer.

Before installation begins, you'll be asked to accept the license for Eclipse like the following screen shown below.

Figure 1:

Click the "Windows 64 bit" link to get started. After doing so, you will see a screen similar to the following.

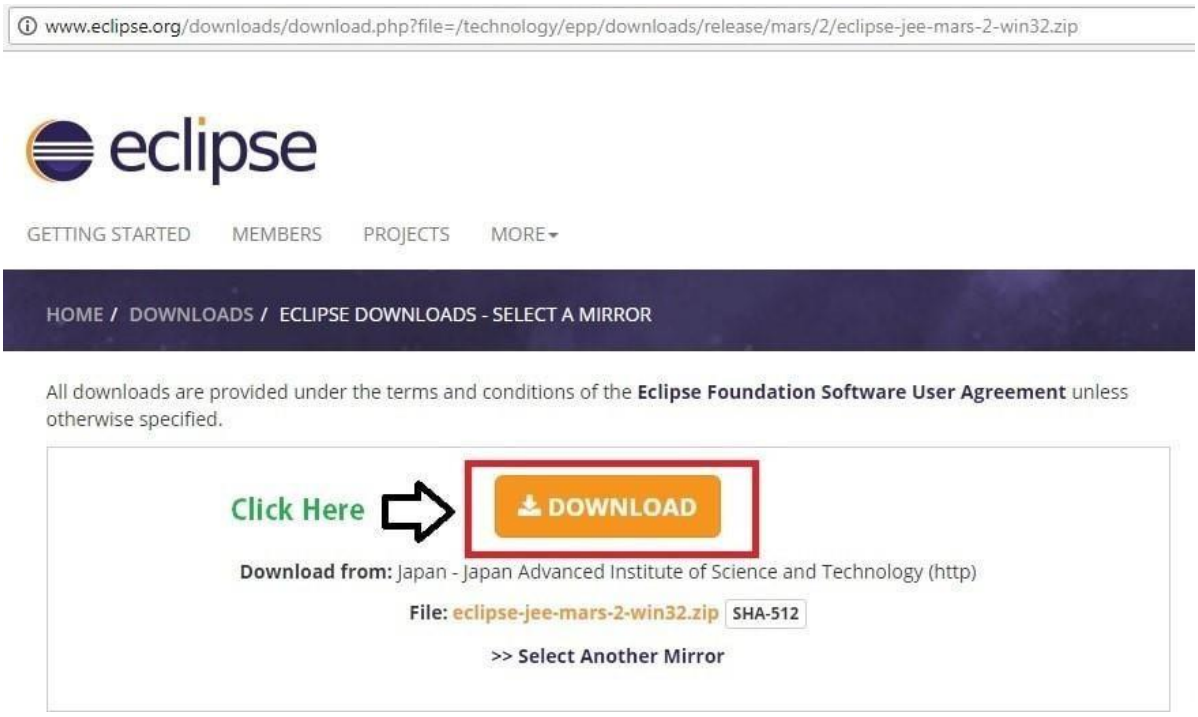


Figure 2:

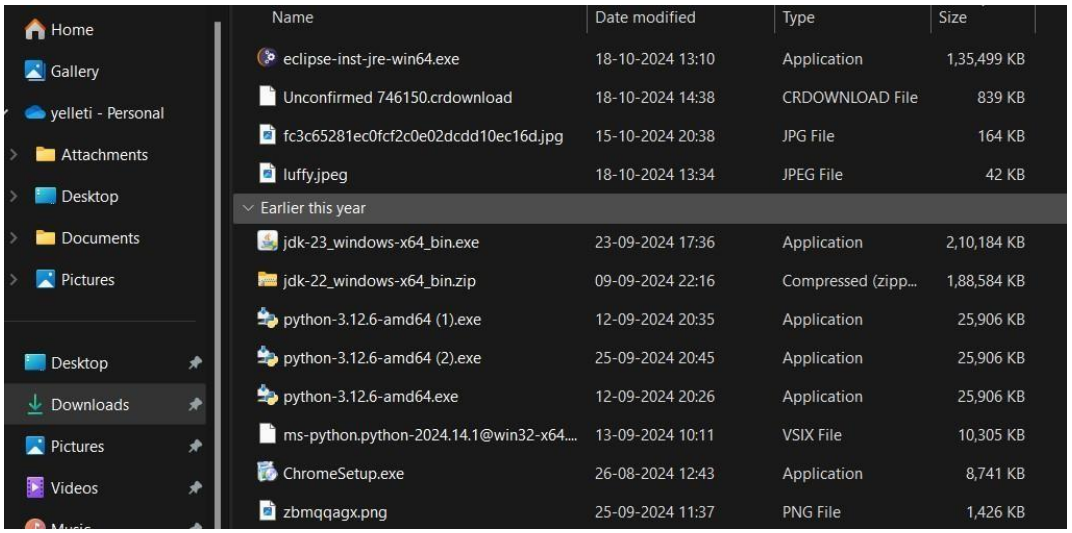


Figure 3:

9.2 Installing Eclipse

After executing the installer, select the “Eclipse IDE for Java Developers” flavor...

The next screen lets you choose a couple of options, which Java version to use, and what directory to install into.

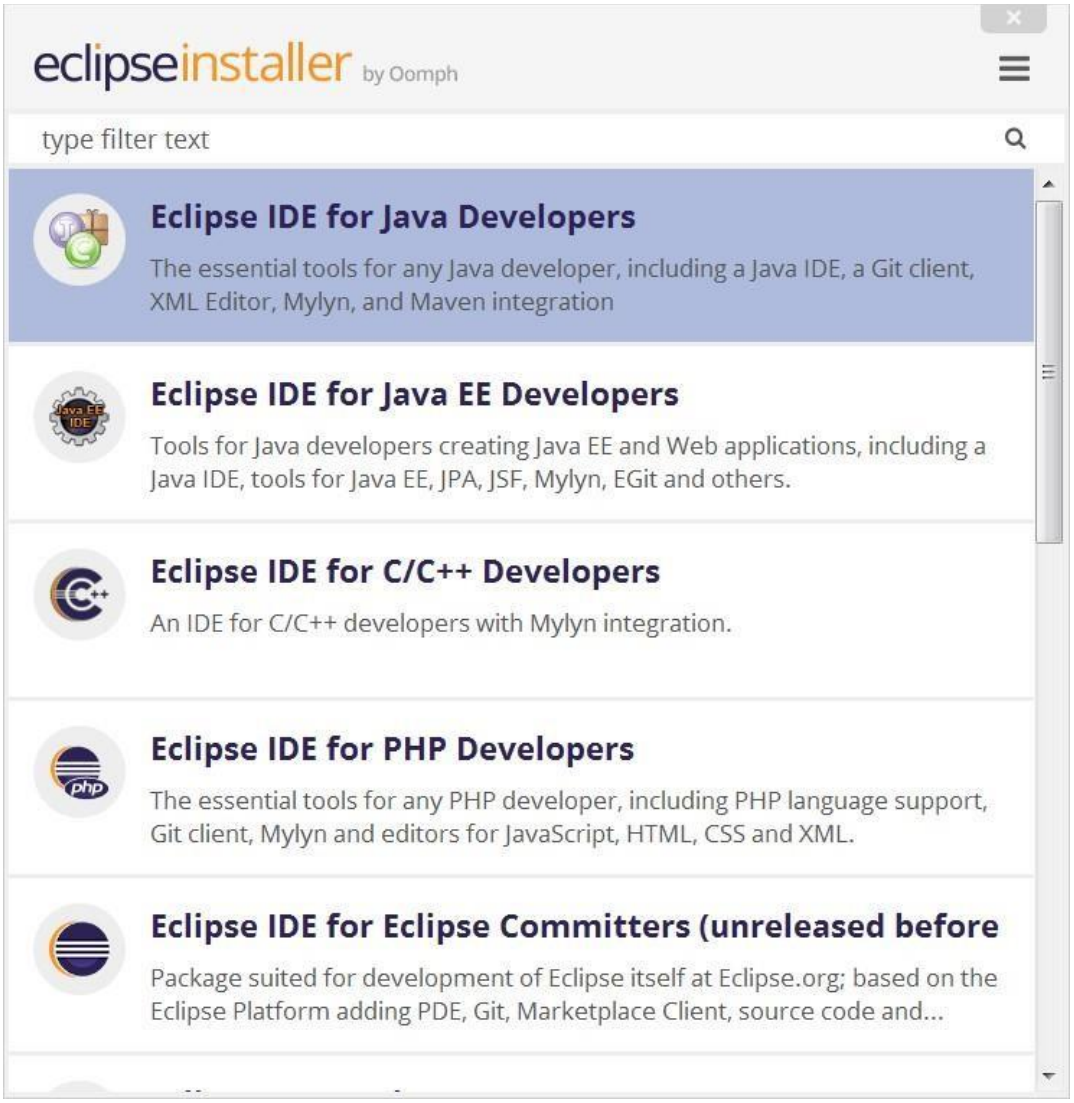

eclipseinstaller by Oomph
type filter text
Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, and Maven integration
Eclipse IDE for Java EE Developers
Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn, EGit and others.
Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.
Eclipse IDE for PHP Developers
The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, HTML, CSS and XML.
Eclipse IDE for Eclipse Committers (unreleased before
Package suited for development of Eclipse itself at Eclipse.org; based on the Eclipse Platform adding PDE, Git, Marketplace Client, source code and...

Figure 4:

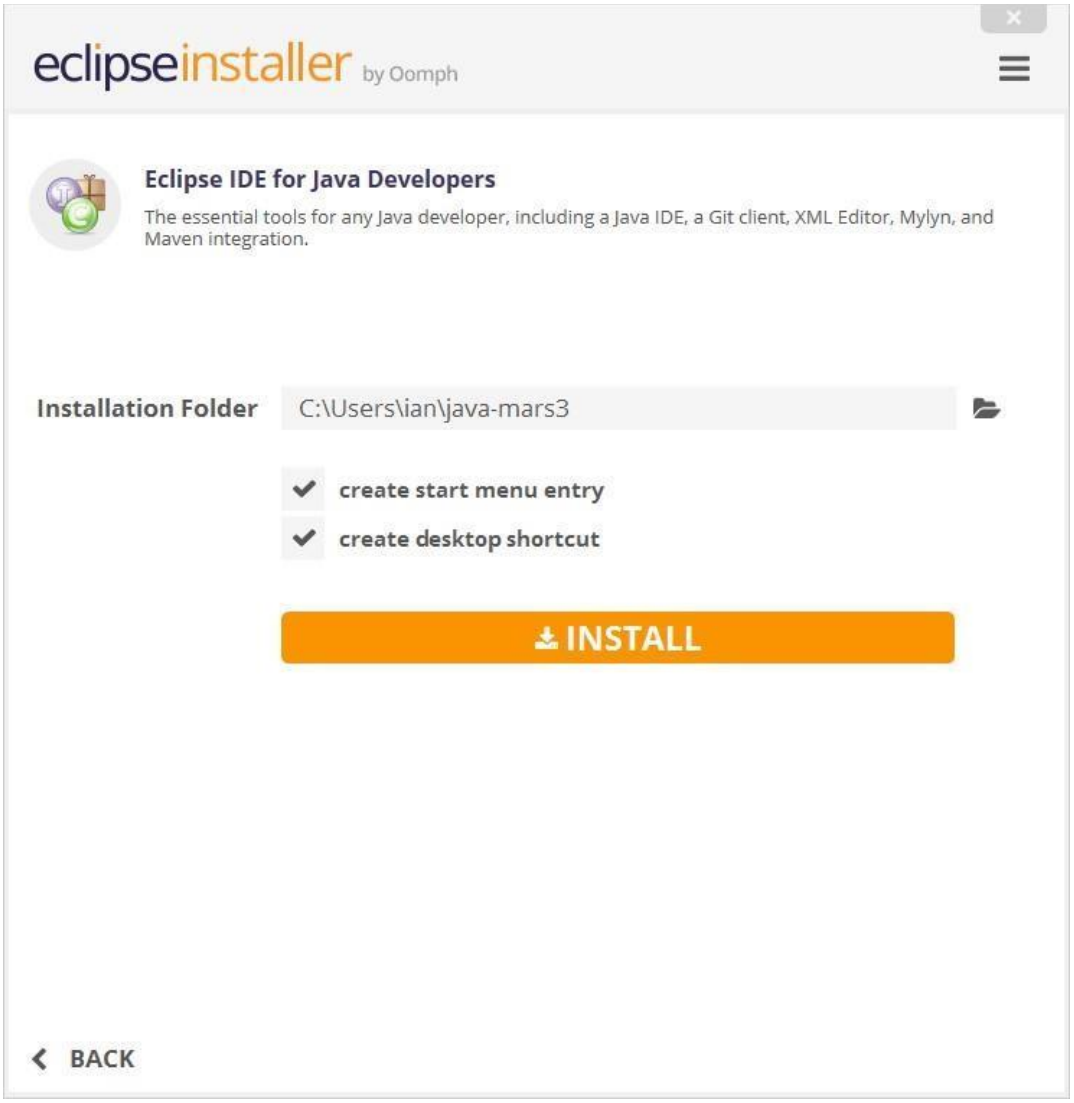

eclipseinstaller by Oomph
Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, and Maven integration.
Installation Folder
C:\Users\ian\java-mars3
create start menu entry
create desktop shortcut
INSTALL
BACK

Figure 5:

Figure 6:
After accepting the license, Eclipse will begin installing and you will see a progress bar

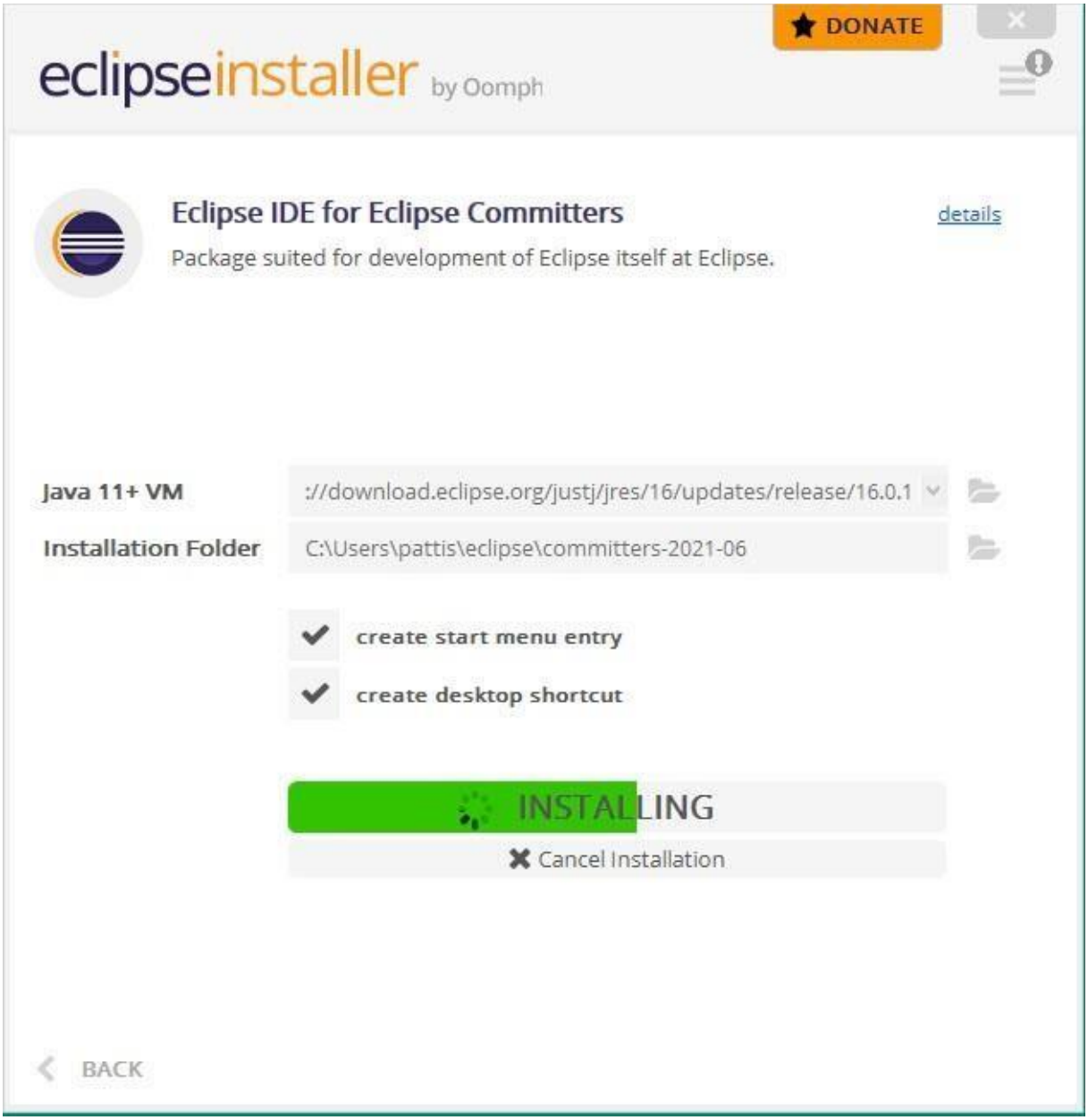


Figure 7:

After the install has finished, you will see a screen similar to the following.

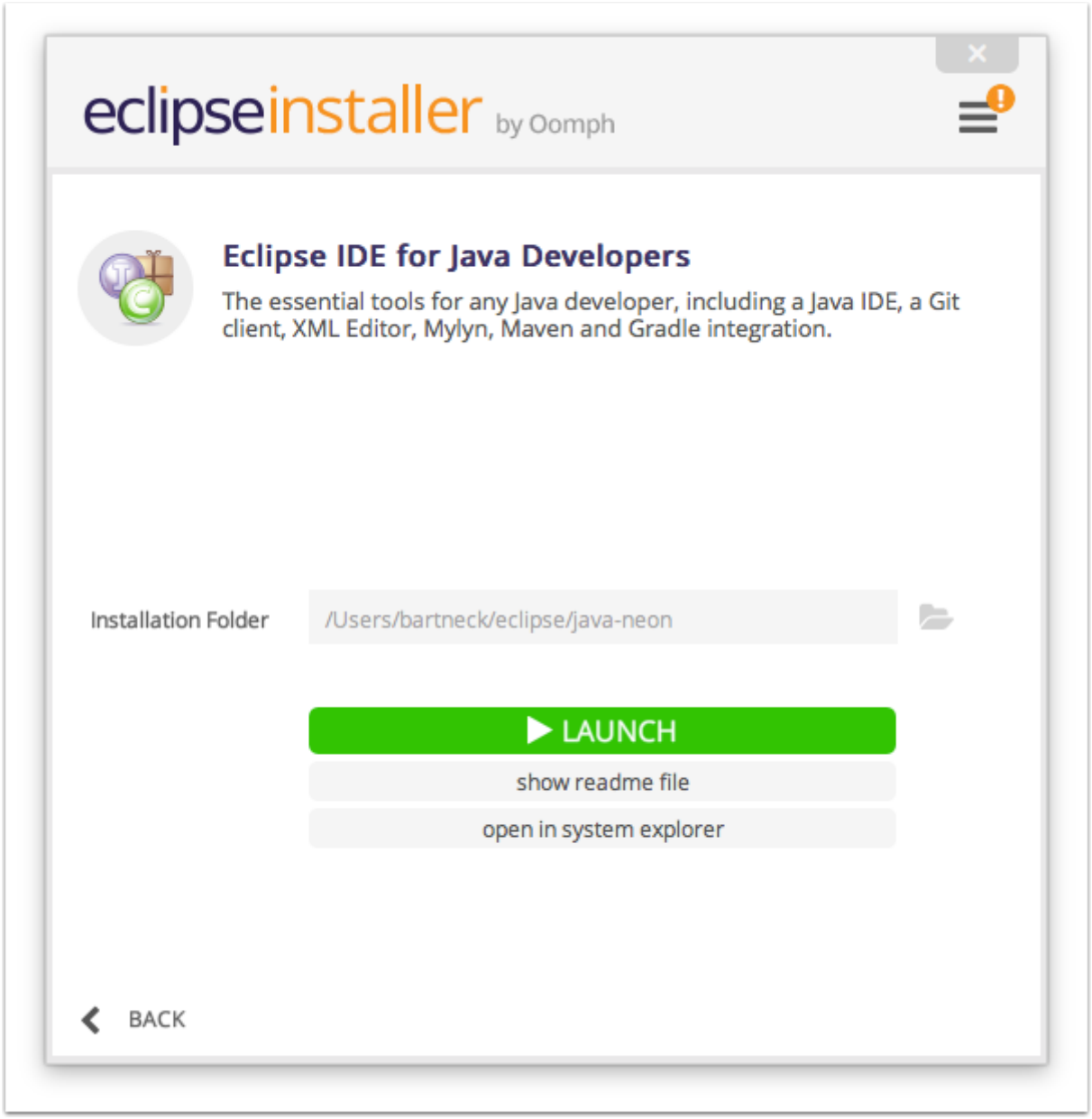


Figure 8:

Creating a Hello World Project

10 Launching Eclipse

After the installation is completed, launch Eclipse for the first time by clicking the "Launch" button...

11 Choosing a Workspace

The first time Eclipse launches, you are asked to decide on a base directory...

12 Creating the Project

To create a Java project, click File → New → Java Project, and name it ”HelloWorld”...
Running the Project
Once the class is created, you can choose to ”Run” or ”Debug” it... **Exporting a Project for Submission**

13 Exporting the Project

In order to submit your project, you will need to export it into a zip file...

Aim : JUNIT Testing in Eclipse Oxygen IDE

Experiment-1

Consider an automated banking application. The user can dial the bank from a personal computer, provide a six-digit password, and follow with a series of keyword commands that activate the banking function.

Test Case Id	Functionality	Test Case Description	Expected Result	Actual Result	Screenshots	Positive/Negative	Remarks
TC001	Dail the bank from PC	Provide the 3 digit area code, 3 digit prefix and 4 digit suffix number.	The user must be able to dail the dail the bank from his/her PC.	valid		Positive	Equivalance Class Testing.
TC002	Dail the bank from PC	Provide the **blank area code**, 3 digit prefix and 4 digit suffix number.	The user must be able to dail the dail the bank from his/her PC.	valid		Positive	BVA Testing
TC003	Dail the bank from PC	Provide the **special characters** in the place of area code, prefix and suffix.	The user should not be allowed to dail the bank from his\her PC.	valid		Negative	Robust Wrost BVA.
TC004	Dail the bank from PC	Provide the blank area code, suffix and prefix.	The user should not be allowed to dail the bank from his\her PC.	valid		Negative	Robust BVA.
TC005	Dail the bank from PC	Provide the 3 digit area code, 3 digit prefix(beginning with 0) and 4 digit suffix number.	The user should not be allowed to dail the bank from his\her PC.	valid		Negative	Equivalance Class Testing.
TC006	Dail the bank from PC	Provide the 3 digit area code, 3 digit prefix(beginning with 1) and 4 digit suffix number.	The user should not be allowed to dail the bank from his\her PC.	valid		Negative	Equivalance Class Testing.
TC007	Password Verification	3 digit prefix and 4 digit suffix number. 2.) Provide the correct password to login into the banking system.	The user should be able to login into banking application successfully.	valid		Positive	Functional Testing
TC008	Password Verification	3 digit prefix and 4 digit suffix number. 2.) Provide the wrong password to login into the banking system.	The user should not be able to login into banking application successfully and presented with the appropriate error message.			Negative	BVA Testing
		3 digit prefix and 4 digit suffix number. 2.) Provide the blank password to login into the banking	The user should not be able to login into banking application successfully and presented with the appropriate error				

Figure 9:

Java Code:

```
package bankapplication ;

public class BankApplication {

    public int pref ( int pr) { if (pr >= 200 && pr
        <= 999) { return 1;
        }
        return 2;
    }

    public int suf ( int s ) { int i = 0
        , x ; while ( s != 0) { x = s ;
        i++;
            s = x / 10;
```

```
            }
            if ( i == 4) {
                return 3;
            }
            return 4;
        }

        public int password( String ps) { if (ps .
            length () == 6) { return 5;
            }
            return 6;
        public int cmds( String cmd) {

            if (cmd. equals ("Check Status")
            }
            return 8;
        }

        public int areacode ( int [ ] ae) { int count = 0;


                for ( int i = 0;              i < ae . length ;   i++) {
                  count++; }

               if ( count == 0      | |          count == 3){

        }


                return 9;
            }
            return 10;
        }
}
```

Test Cases

```
package pack ;

import static org . junit . Assert .*;

import org . junit . Test ;

public class AtmTest {
        @Test
```

```
        public void test () { Atm a1 = new Atm();
                int p1 = a1 . withdraw (300);
                 assertEquals (700 , p1 );
        }
}
```

Test Case 1

```
package pack ;

import static org . junit . Assert .*;

import org . junit . Test ;

 public class DepositTest { @Test public void
        test () { Atm a1 = new Atm(); int p2 = a1
        . deposit (1000);
                assertEquals (1700 , p2 );
        }
}
```

Test Case 2

```
package pack ;

 import static org . junit . Assert .*;

 import org . junit . Test ;

public class PinTest { public void test ()
        { Atm a1 = new Atm(); int p4 = a1 .
        pin (5555);
                assertEquals (1 , p4 );
        }
}
```

Test Case 3

```
package pack ;

import static org . junit . Assert .*;

import org . junit . Test ;
```

```
public class PinTest { @Test public void
        test () { Atm a1 = new Atm(); int p4
        = a1 . pin (5555);
                assertEquals (1 , p4 );
        }
}
```

Test Suite

```
package atm;

import org . junit . runner .RunWith;

 import org . junit . runners . Suite ;

 import org . junit . runners . Suite . SuiteClasses ;

@RunWith( Suite . class )
@SuiteClasses ({ AtmTest . class , T2. class , T3. class , T4. class }) public class AllTests
{
}
```

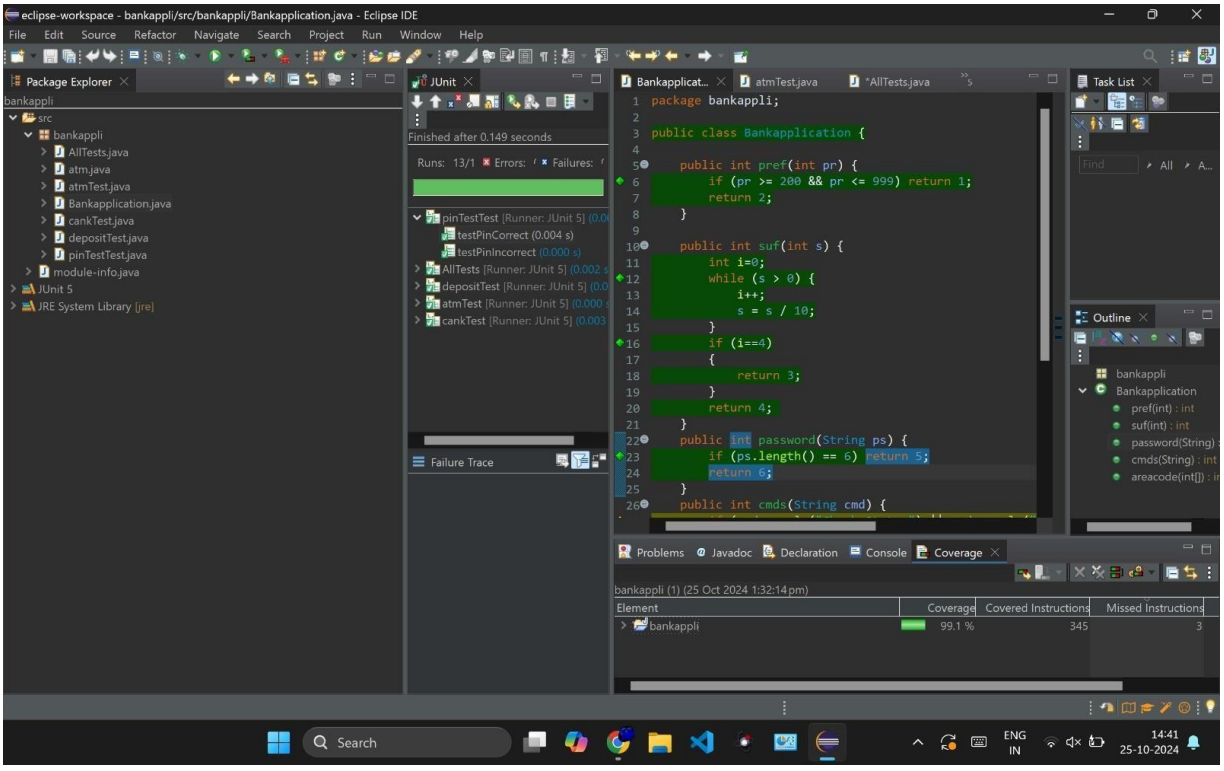


Figure 10:

Experiment-2

Consider an automated banking application. The user can dial the bank from a personal computer, provide a six-digit password, and follow with a series of keyword commands

that activate the banking function. Design test cases for Boundary Value Analysis (BVA) and Equivalence Class Testing.

Test Case Id	Functionality	Test Case Description	Expected Result	Actual Result	Screenshots	Positive/Negative	Remarks
TC001	Dail the bank from PC	Provide the 3 digit area code, 3 digit prefix and 4 digit suffix number.	The user must be able to dail the dail the bank from his/her PC.	valid		Positive	Equivalance Class Testing.
TC002	Dail the bank from PC	Provide the **blank area code**, 3 digit prefix and 4 digit suffix number.	The user must be able to dail the dail the bank from his/her PC.	valid		Positive	BVA Testing
TC003	Dail the bank from PC	Provide the **special characters** in the place of area code, prefix and suffix.	The user should not be allowed to dail the bank from his\her PC.	valid		Negative	Robust Wrost BVA.
TC004	Dail the bank from PC	Provide the blank area code, suffix and prefix.	The user should not be allowed to dail the bank from his\her PC.	valid		Negative	Robust BVA.
TC005	Dail the bank from PC	Provide the 3 digit area code, 3 digit prefix(beginning with 0) and 4 digit suffix number.	The user should not be allowed to dail the bank from his\her PC.	valid		Negative	Equivalance Class Testing.
TC006	Dail the bank from PC	Provide the 3 digit area code, 3 digit prefix(beginning with 1) and 4 digit suffix number.	The user should not be allowed to dail the bank from his\her PC.	valid		Negative	Equivalance Class Testing.
TC007	Password Verification	3 digit prefix and 4 digit suffix number. 2.) Provide the correct password to login into the banking system.	The user should be able to login into banking application successfully.	valid		Positive	Functional Testing
TC008	Password Verification	3 digit prefix and 4 digit suffix number. 2.) Provide the wrong password to login into the banking system.	The user should not be able to login into banking application successfully and presented with the appropriate error message.			Negative	BVA Testing
		3 digit prefix and 4 digit suffix number. 2.) Provide the blank password to login into the banking	The user should not be able to login into banking application successfully and presented with the appropriate error				

Figure 11:

Java Code:

```
package bankappli;


public class BankApplication {


  public int pref(int pr) {
    if (pr >= 200 && pr <= 999) {
      return 1;
    }
    return 2;
  }


  public int suf(int s) {
```

```
        int i = 0, x;
        while (s != 0) {
            x = s;
            i++;
            s = x / 10;
        }
        if (i == 4) {
            return 3;
        }
        return 4;
    }

    public int password(String ps) {
        if (ps.length() == 6) {
            return 5;
        }
        return 6;
    }

    public int cmds(String cmd) {
        if (cmd.equals("Check Status") || cmd.equals("Deposit") || cmd.equals("Withdraw"))
{
            return 7;
        }
        return 8;
    }
```

```
  public int areacode(int[] ae) {
    int count = 0;
    for (int i = 0; i < ae.length; i++) {
      count++;
    }
    if (count == 0 || count == 3) {
      return 9;
    }
    return 10;
  }
}
```

Test Case 1:

```
package pack ;

import static org . junit . Assert .*;

import org . junit . Test ;

public class AtmTest {
        @Test
        public void test () { Atm a1 = new Atm();
                int p1 = a1 . withdraw (300);
                 assertEquals (700 , p1 );
        }
}
```

Test Case 2:

```
package pack ;

import static org . junit . Assert .*;

import org . junit . Test ;

public class DepositTest {
```

```
        @Test public void test () { Atm a1 = new
        Atm(); int p2 = a1 . deposit (1000);
                assertEquals (1700 , p2 );
        }
}
```

Test Case 3:

```
package pack ;

import static org . junit . Assert .*;

import org . junit . Test ;

public class PinTest { public void test ()
        { Atm a1 = new Atm(); int p4 = a1 .
        pin (5555);
                assertEquals (1 , p4 );
        }
}
```

Test Suite

```
package atm;

import org . junit . runner .RunWith;

import org . junit . runners . Suite ;

import org . junit . runners . Suite . SuiteClasses ;

@RunWith( Suite . class )
@SuiteClasses ({ AtmTest . class , T2. class , T3. class , T4. class }) public class AllTests
{
```

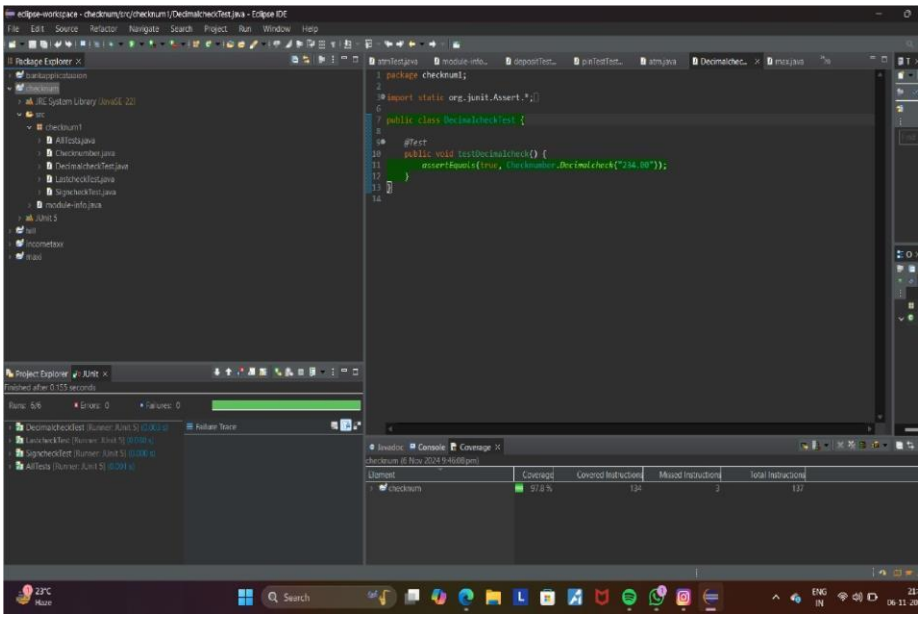
}

Figure 12:

Experiment-3

The application is required to validate a number according to the following rules:

1. A number can start with an optional sign .
2. The optional sign can be followed by any number of digits.
3. The digits can be optionally followed by a decimal point, represented by a period.
4. If there is a decimal point, there should be exactly two digits after the decimalpoint.
5. The number, whether or not it has a decimal point, should be terminated by ablank space.



Test Case Id	Functionality	Test Case Description	Expected Result	Actual Result	Screenshots	Positive/Negative	Remarks
TC001	Validate Number	Provide the number "+1234 ".	The application should not accept this number because the SPACES are not allowed at the end of the number.	valid		Negative	decision table
TC002	Validate Number	Provide the number "+ " and "- ".	The application should accept this number because it's a valid number. And also shouldn't end with SPACES.	invalid		Positive	decision table
TC003	Validate Number	Provide the number "1234.23 ".	The application should not accept this number because the SPACES are not allowed at the end of the number.	valid		positive	decision table
TC004	Validate Number	Provide the number "1234 ".	The application should not accept this number because the SPACES are not allowed at the end of the number.	valid		positive	decision table
TC005	Validate Number	Provide the number "*1234".	The application should not accept this number because the special charater * is not allowed at the beginning of the number.	valid		positive	decision table
TC006	Validate Number	Provide SPACES in the place of number.	The application should not accept this number because the SPACES are not allowed.	valid		positive	decision table
TC007	Validate Number	Provide just "+" and "-" in the place of number.	The application should not accept this number because only signs are not allowed in the place number.	valid		positive	decision table
TC008	Validate Number	Provide number "+. "	The application should not accept this number because numer shouldn't end by decimal point. And also number shoudnt end with SPACES.	valid		positive	decision table
TC009	Validate Number	Provide number "+1234.1"	The application should not accept this number because two digits should be followed after decimal point.	valid		positive	decision table
TC010	Validate Number	Provide number "+1234.12"	The application should accept this number because it's a valid number.	valid		Positive	decision table
TC011	Validate Number	Provide the number "+1234.56*"	The application should not accept this number because the number is getting ended with a special character.	valid		positive	decision table
			The application should not accept this				

Figure 13:

Java Code:

JAVACODE:

```
public c l a s s Checknumber {

Scanner s c= new Scanner ( System . i n ) ;

int c =1;

s t a t i c boolean a=f a l s e ;

s t a t i c boolean l=f a l s e ;

public Boolean S i g n c h e c k ( S t r i n g s t r ) {

char [ ] b= s t r . toCharArray ( ) ;

i f ( b[0]== '+ ' | | b[0]== '– ' )

{

a=true ;

}

return a ;

}

public b o o l e a n L a s t c h e c k ( S t r i n g s t r ) {
```

```
boolean l=f a l s e ;
char [ ] b= s t r . toCharArray ( ) ;
int l e n=b . l e n g t h ;
i f ( b [ len −1]== ' ' )
{
l=true ;
}
return l ;
39
}
public int CheckDigit ( int num) {
i f (num==(int )num)
c =0;
e l s e
c =1;
return c ;
}
public s t a t i c b o o l e a n D e c i m a l c h e c k ( S t r i n g s t r ) {
char [ ] b= s t r . toCharArray ( ) ;
int l e n=b . l e n g t h ;
for ( int i =0; i <l e n ; i ++) {
System . out . p r i n t l n ( b [ i ] ) ;
i f ( b [ i ]== ' . ' )
{
i f ( b [ i +1]== ' 0 ' ) {
i f ( b [ i +2]== ' 0 ' ) {
```

```
l=true ;
}
}
}
}
return l ;
}
public s t a t i c void main ( S t r i n g [ ] a r g s ){
Decimalcheck ( " 2 3 4 . 0 0 " ) ;
}
}
```

Test Case 1:

```
package number ;
import s t a t i c org . j u n i t . A s s e r t . * ;
import org . j u n i t . Test ;
public c l a s s DecimalcheckTest {
@Test
public void t e s t ( ) {
Checknumberch = new Checknumber ( ) ;
a s s e r t E q u a l s ( true , ch . Decimalcheck ( " 2 3 4 . 0 0 " ) ) ;
}
}
```

Test Case 1:

```
package number ;
import static org . junit . Assert .*;
```

```
import org . junit . Test ; public class
DecimalcheckTest {

  @Test public void test () {
    Checknumberch = new Checknumber (); assertEquals (true ,
  ch . Decimalcheck (”234.00” )); }

}
```

Test Case 2:

```
package number ;

 import static org . junit . Assert .∗;

import org . junit . Test ; public class
LastcheckTest {

  @Test
  public void test () {
    Checknumberch = new Checknumber (); assertEquals
  (true , ch . Lastcheck (”234 ” )); }

}
```

Test Case 3:

```
package number ;

 import static org . junit . Assert .∗;

import org . junit . Test ; public class
SigncheckTest {

  @Test
  public void test () {
    Checknumberch = new Checknumber (); assertEquals
  (true , ch . Signcheck (”−234” )); }

}
```

Test Suite:

```
package numbervalid ;
```

```
import org . junit . runner .RunWith;

 import org . junit . runners . Suite ;

import org . junit . runners . Suite . SuiteClasses ;

@RunWith( Suite . class )
@SuiteClasses ({ T1. class , T2. class , T3. class , T4. class }) public class AllTests {

}
```

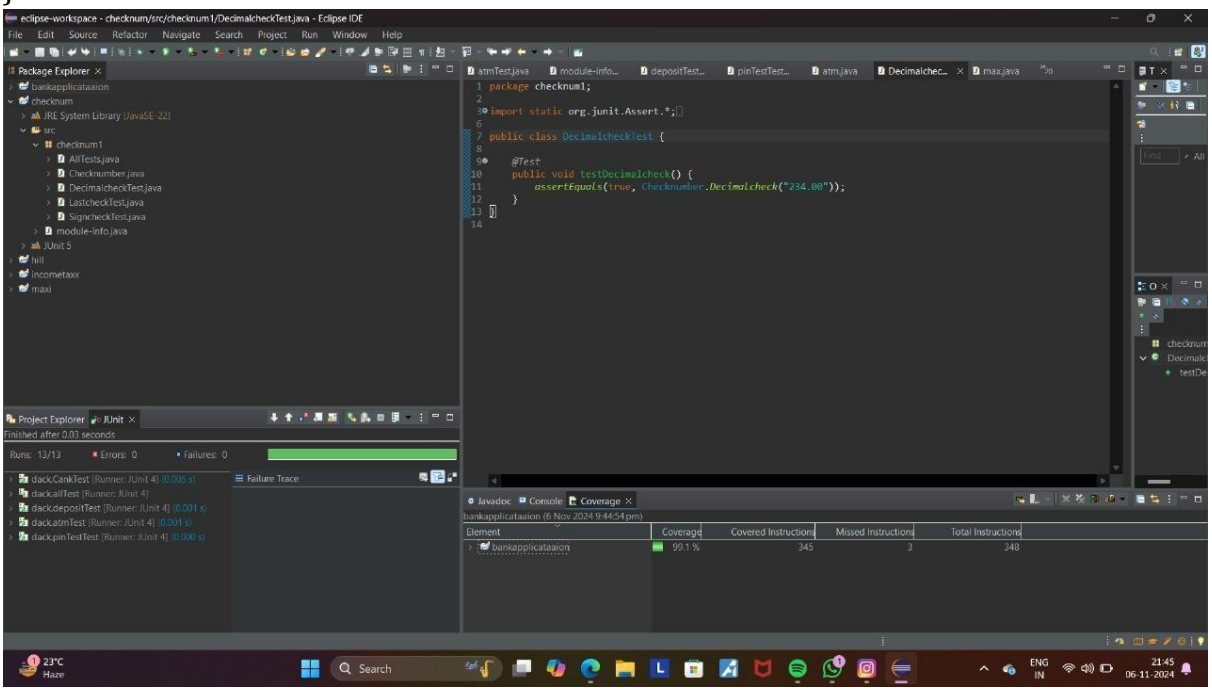


Figure 14:

Experiment-4

Generate test cases using Black box testing technique to Calculate Standard Deduction on Taxable Income. The standard deduction is higher for tax payers who are 65 or older or blind. Use the method given below to calculate tax.

1. **Filing Status:** The basic standard deduction for various filing statuses are:
    - **Single:** $4,750
    - **Married, filing a joint return:** $9,500
    - **Married, filing a separate return:** $7,000
2. **Married Filing Separately:** If a married couple is filing separate returns and one spouse is not taking the standard deduction, the other spouse also is not eligible for the standard deduction.

3. **Additional Deduction for Age:** An additional $1,000 is allowed as a standard deduction if either the filer or the spouse is 65 years or older (the latter case is applicable when the filer is married and filing jointly).

4. **Additional Deduction for Blindness:** An additional $1,000 is allowed as a standard deduction if either the filer or the spouse is blind (this is also applicable when the filing status is married).

Test Case Id	Functionality	Test Case Description	Expected Result	Actual Result	Screenshots	Positive/Negativ	Remarks
TC001	Single Tax Calculation	Enter the filler age greater than 65 and not blind.	5750 is the total tax must be deducted.	valid		positive	blackbox
TC002	Single Tax Calculation	Enter the filler age less than 65 and not blind.	4750 is the total tax must be deducted.	valid		positive	blackbox
TC003	Single Tax Calculation	Enter the filler age greater than 65 and blind.	7750 is the total tax must be deducted.	valid		positive	blackbox
TC004	Single Tax Calculation	Enter the filler age less than 65 and blind.	6750 is the total tax must be deducted.	valid		positive	blackbox
TC005	Married - Separate Filling	Enter the filler age greater than 65 and not blind.	8000 is the total tax must be deducted.	valid		positive	blackbox
TC006	Married - Separate Filling	Enter the filler age less than 65 and not blind.	7000 is the total tax must be deducted.	valid		positive	blackbox
TC007	Married - Separate Filling	Enter the filler age greater than 65 and blind.	9000 is the total tax must be deducted.	valid		positive	blackbox
TC008	Married - Separate Filling	Enter the filler age less than 65 and blind.	8000 is the total tax must be deducted.	valid		positive	blackbox
TC009	Married - Joint Filing	Enter the filler age less than 65, spouse age less than 65. Both filler and spouse are not blind.	9500 is the total tax must be deducted.	valid		positive	blackbox
TC010	Married - Joint Filing	Enter the filler age less than 65, spouse age less than 65. Filler is blind and spouse is not blind.	10500 is the total tax must be deducted.	valid		positive	blackbox
TC011	Married - Joint Filing	Enter the filler age less than 65, spouse age less than 65. Filler is not blind and spouse is blind.	10500 is the total tax must be deducted.	valid		positive	blackbox
TC012	Married - Joint Filing	Enter the filler age less than 65, spouse age less than 65. Both filler and spouse are blind.	11500 is the total tax must be deducted.	valid		positive	blackbox
TC013	Married - Joint Filing	Enter the filler age less than 65, spouse age greater than 65. Both filler and spouse are not	10500 is the total tax must be deducted.	valid		positive	blackbox
		Enter the filler age less than 65, spouse age greater than 65.	11500 is the total tax must be				

Figure 15:

Java Code:

```
package income ;
 public class Incometax {
  int i =20000;
  public int status ( String a)
  { if (a== " single ")
      return i –4750;
    return i ;

  }
public int married ( String ma, Stringj , int mage                    , int fage                , String f , String


    if (ma=="married"&&j==" joint ")
    { if ( f=="blind" | |m=="blind")
      { if (mage>=65||fage >=65)
          return i –11500;
        return i –10500;
```

```
    } if (mage>=65||fage >=65)
      return i −10500;

  } return i −9500;
 }
 public int seperate ( String ma, Strings , Stringf , String m)
 { if (ma==”married” )

  { if ( s==”seperate”)
    { if ( f==”yes”)

    { if (m==”yes”)
        return i −7000;
    }



  }
    }
  return i ;
  } }
```

Test case 1:

```
package income ;

 import static org . junit . Assert .∗;

import org . junit . Test ; public class T1 {

  @Test
  public void test () { Incometax t=new
    Incometax (); int r=t . status (” single ”
    ); assertEquals (20000−4750,r );
  }
}
```

Test case 2:

```
package income ;
 import static org . junit . Assert .∗;

 import org . junit . Test ; public class T2 {
```

```
@Test public void test () {
    Incometax t=new Incometax ();
                        int r=t . married (”married” ,” joint ” ,75 ,55 ,”blind” ,”blind” );

assertEquals (8500 , r );
  }}
```

Test case 3:

```
package income ;

import static org . junit . Assert .∗;

 import org . junit . Test ; public class T3 {

  @Test
  public void test () { Incometax t=new
    Incometax ();
    int r=t . seperate (”married” ,”seperate” ,”yes” ,”yes” ); assertEquals (20000−7000,r );
  }
  }
```

Test Suite:

```
package income ;
import org . junit . runner .RunWith;
import org . junit . runners . Suite ;
import org . junit . runners . Suite . SuiteClasses ;
@RunWith( Suite . class )
@SuiteClasses ({ T1. class , T2. class , T3. class }) public class AllTests {
}
```

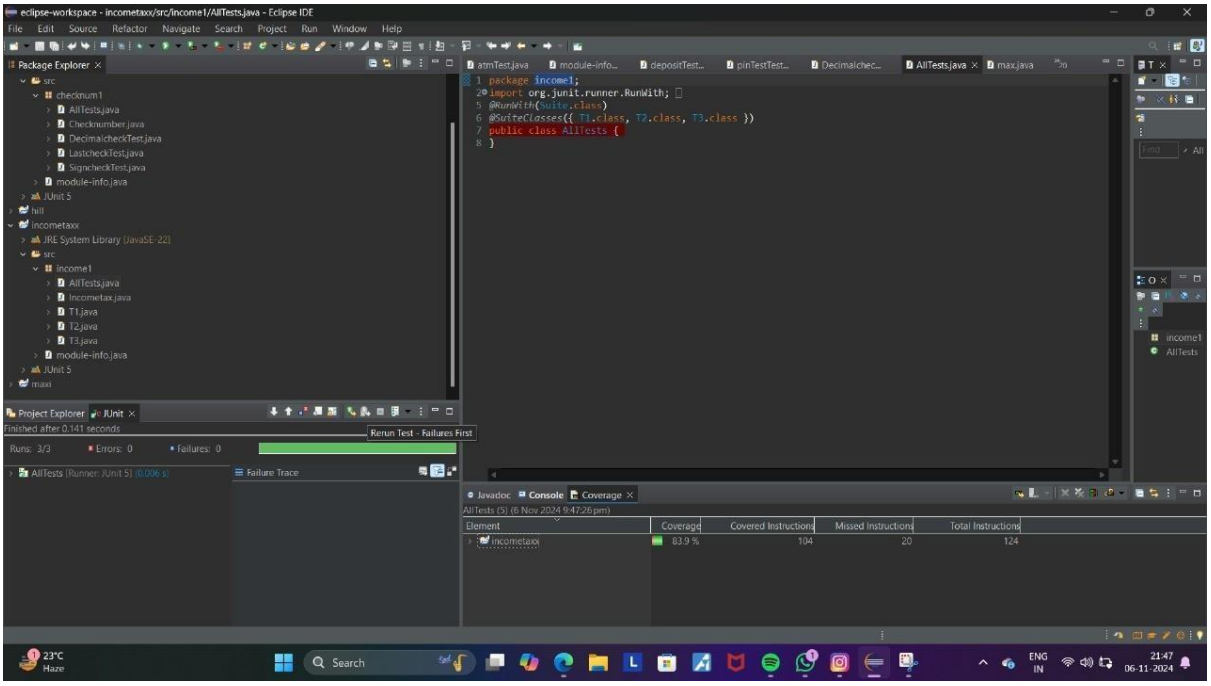

eclipse-workspace - incometaxx/src/income1/AllTests.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
src
checknum1
AllTests.java
Checknumber.java
DecimalcheckTest.java
LastcheckTest.java
SigncheckTest.java
module-info.java
JUnit 5
hill
incometaxx
JRE System Library [JavaSE-22]
src
income1
AllTests.java
Incometax.java
T1.java
T2.java
T3.java
module-info.java
JUnit 5
maxi
atmTest.java
module-info...
depositTest...
pinTestTest...
Decimalchec...
AllTests.java
max.java
1 package income1;
2 import org.junit.runner.RunWith;
5 @RunWith(Suite.class)
6 @SuiteClasses({ T1.class, T2.class, T3.class })
7 public class AllTests {
8 }
Project Explorer
JUnit
Finished after 0.141 seconds
Runs: 3/3
Errors: 0
Failures: 0
Rerun Test - Failures First
AllTests [Runner: JUnit 5] (0.006 s)
Failure Trace
Javadoc
Console
Coverage
AllTests (5) (6 Nov 2024 9:47:26 pm)
Element
Coverage
Covered Instructions
Missed Instructions
Total Instructions
incometaxx
83.9 %
104
20
124
23°C
Haze
Search
ENG
IN
21:47
06-11-2024

Figure 16: