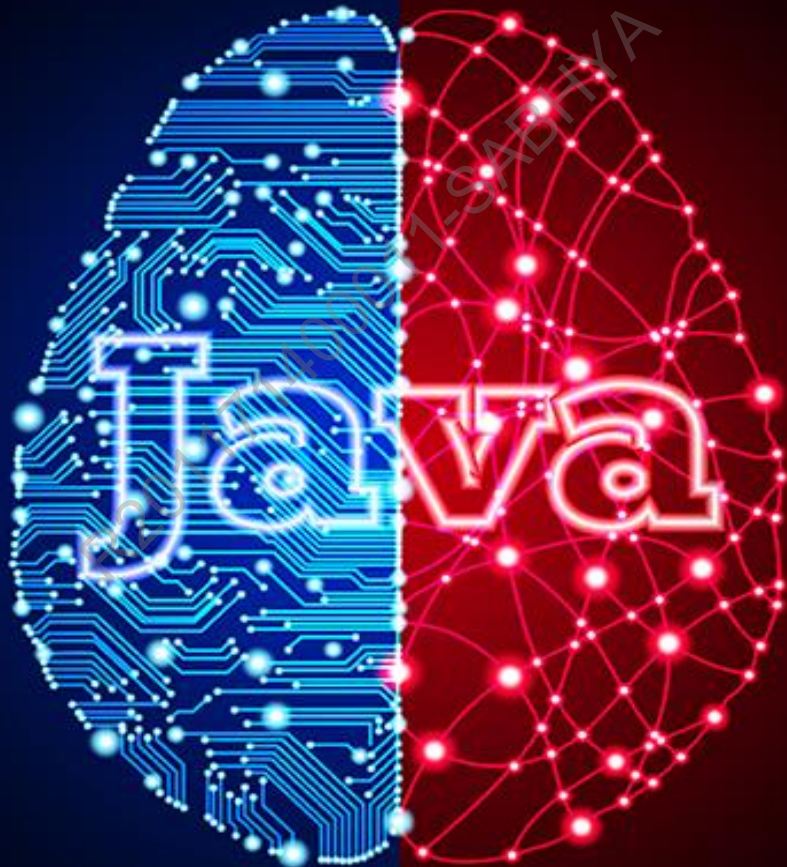


PROGRAMMING IN JAVA

Activity Book



NIIT

R201171400011-SABHYA

Programming in Java

Activity Book

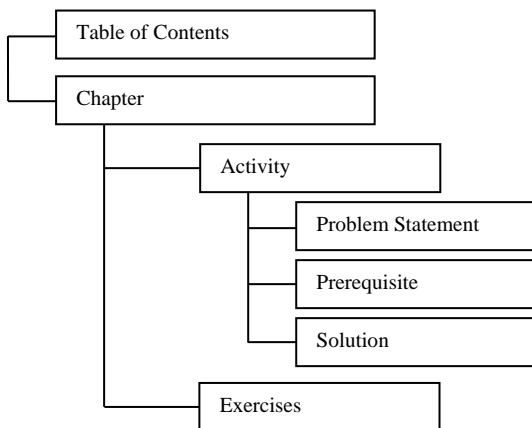
Trademark Acknowledgements

All products are registered trademarks of their respective organizations.
All software is used for educational purposes only.

Programming in Java/AB/18-M08-V1.0
Copyright ©NIIT. All rights reserved.

No part of this publication may be reproduced, stored in retrieval system or transmitted in any form, or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the publisher.

COURSE DESIGN – ACTIVITY BOOK



R201171400011-SABHYA

Table of Contents

Chapter 1 – Designing a User Interface and Handling Events

Exercises -----	1.2
Exercise 1 -----	1.2
Exercise 2 -----	1.3

Chapter 2 – Implementing Inner Classes

Activity 2.1: Creating Inner Classes -----	2.2
Problem Statement-----	2.2
Solution -----	2.2
Exercises -----	2.4
Exercise 1 -----	2.4

Chapter 3 – Working with Localization

Activity 3.1: Implementing Localization -----	3.2
Problem Statement-----	3.2
Solution -----	3.2
Exercises -----	3.4
Exercise 1 -----	3.4
Exercise 2 -----	3.4

Chapter 4 – Working with Generics

Activity 4.1: Working with Generics -----	4.2
Problem Statement-----	4.2
Solution -----	4.2
Exercises -----	4.6
Exercise 1 -----	4.6

Chapter 5 – Working with Collections

Activity 5.1: Working with Collections	5.2
Problem Statement	5.2
Solution	5.2
Exercises	5.9
Exercise 1	5.9

Chapter 6 – Working with Threads

Activity 6.1: Implementing Threads in Java	6.2
Problem Statement	6.2
Solution	6.2
Exercises	6.5
Exercise 1	6.5
Exercise 2	6.5

Chapter 7 – Implementing Thread Synchronization and Concurrency

Activity 7.1: Implementing Synchronization	7.2
Problem Statement	7.2
Solution	7.2
Exercises	7.6
Exercise 1	7.6
Exercise 2	7.6

Chapter 8 – Working with NIO Classes and Interfaces

Activity 8.1: Performing File Operations Using NIO	8.2
Problem Statement	8.2
Solution	8.2
Exercises	8.7
Exercise 1	8.7

Chapter 9 – Introduction to JDBC

Activity 9.1: Creating a JDBC Application to Query a Database	9.2
Problem Statement	9.2
Solution	9.2
Exercises	9.5
Exercise 1	9.5

Chapter 10 – Creating Applications Using Advanced Features of JDBC

Activity 10.1: Creating an Application that Uses the PreparedStatement Object	10.2
Problem Statement	10.2
Solution	10.3
Activity 10.2: Creating an Application to Determine the Structure of a Table	10.9
Problem Statement	10.9
Solution	10.9
Exercises	10.12
Exercise 1	10.12



NIIT

R201171400011-SABHYA

Designing a User Interface and Handling Events

CHAPTER 1

Exercises

Exercise 1

Mark has been assigned a task to design the complaint form, as shown in the following figure.

Employee Name: _____	Employee ID: _____
Sex: _____ Male _____ Female	
Email: _____	Contact No.: _____
Department: _____	
Name of Immediate Supervisor: _____	
Problem Category: _____ Infrastructure _____ Food _____ Cab _____ Work Environment	
Please describe the details of your grievances: _____ _____	
Please list the settlement you are requesting: _____ _____	

The Complaint Form

For this, he needs to identify the best suitable components and write a program for the same. Help Mark to achieve the preceding requirement.



Note

You need to create the class of exercise 1 inside the package, **chapter01**, in the application, **Exercises**.

Exercise 2

Create a GUI application that implements a simple arithmetic calculator. The application must have two input text boxes and an additional text box that will be read-only. In addition, the application must have command buttons for arithmetic functions, such as addition, subtraction, multiplication, and division.

The user will enter two numbers in the input text boxes and will click any arithmetic function command button. Once the button is clicked, the result should be displayed in the third text box.



Note

*You need to create the classes of exercise 4 inside the package, **chapter01**, in the application **Exercises**.*

R201171400011-SABHYA

The background of the entire page is a light blue network diagram. It consists of numerous small circular nodes connected by thin, light blue lines, forming a complex web-like structure. The nodes are distributed across the page, with some clusters being denser than others. The overall effect is a technical, digital aesthetic.

NIIT

R201171400011-SABHYA

Implementing Inner Classes

CHAPTER 2



Activity 2.1: Creating Inner Classes

Problem Statement

Mellos Inc. is a software development company. The company has received an assignment to develop an aptitude assessment application. The management of the company has assigned this task to Mike, the Senior Developer of the company. In the initial phase, Mike needs to create a class that accepts the candidate details, such as name, qualification, and age. In addition, he needs to validate the age of the candidate who can appear for the test. The age of the candidate must range between 21 and 25. Further, Mike needs to logically group the class, so that each class provides its own functionality. Help Mike in achieving the preceding tasks.

Solution

To achieve the preceding requirements, Mike needs to perform the following steps:

1. Create a Java application, **InnerClassDemo**.
2. Create a package, **details**, in the Java application, **InnerClassDemo**.
3. Create a class, **Candidates**, inside the package, **details**.
4. Replace the code in the **Candidates.java** file with the following code:

```
package details;
import java.util.*;
public class Candidates {
    private String name, qualification, status;
    private int age;

    public void getCandidatesDetails(){
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter your name:");
        name=sc.next();
        System.out.println("Enter your qualification:");
        qualification=sc.next();
        System.out.println("Enter your age:");
        age=sc.nextInt();
        AgeValidation obj=new AgeValidation();
        obj.validateAge();
    }
    private class AgeValidation{
        public void validateAge(){
            if(age>=21 && age<=25){
                status="selected";
            }
            else{
                status="rejected";
            }
        }
    }
}
```

```

    public void printCandidatesDetails(){
        System.out.println("Name: "+name);
        System.out.println("Qualification: "+qualification);
        System.out.println("Age: "+age);
        System.out.println("Selection Status: "+status);
    }
    public static void main(String args[]){
        Candidates candidate=new Candidates();
        candidate.getCandidatesDetails();
        candidate.printCandidatesDetails();
    }
}

```

5. Compile and execute the **InnerClassDemo** Java application.

After executing the application, the following output is displayed:

Enter your name:

Once you provide the name as Andrew, the following output is displayed:

Enter your qualification:

Once you provide the qualification as MBA, the following output is displayed:

Enter your age:

Once you provide the age as 23, the following output is displayed:

```

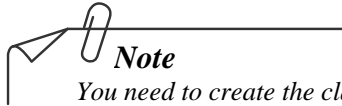
Name: Andrew
Qualification: MBA
Age: 23
Selection Status: selected

```

Exercises

Exercise 1

Write a program to display a frame. In addition, you need to handle the movement of the mouse by using an anonymous inner class, so that the position of the mouse pointer is displayed by a label inside the frame.



Note

You need to create the classes of the exercise 1 inside the package, **chapter02**, inside the application, **Exercises**.

R201171400011-SABHYA

The background of the entire page is a light blue network diagram. It consists of numerous small circular nodes connected by thin, light blue lines, forming a complex web-like structure. The nodes are distributed across the page, with some clusters being denser than others. The overall effect is a technical, digital aesthetic.

NIIT

R201171400011-SABHYA

Working with Localization

CHAPTER 3



Activity 3.1: Implementing Localization

Problem Statement

You need to develop a puzzle game application that supports various languages, such as French and English, so that it can be used in different countries, such as France and US. In the initial phase, you only need to design a localized menu for the game application.

Prerequisite

To perform the activity, you need to use the **Puzzle.txt** and **GameWindow.txt** files.

Solution

To achieve the preceding requirements, you need to perform the following steps:

1. Create a Java application, **LocalizationApp**.
2. Create a package, **localize**, in the application, **LocalizationApp**.
3. Create a class, **Puzzle**, inside the package, **localize**.
4. Open the **Puzzle.txt** file.
5. Replace the code in the **Puzzle.java** file with the code in the **Puzzle.txt** file.
6. Create a class, **GameWindow**, inside the package, **localize**.
7. Open the **GameWindow.txt** file.
8. Replace the code in the **GameWindow.java** file with the code in the **GameWindow.txt** file.
9. Add the following code after the `//English Language` comment:

```
currentLocale= new Locale("en", "US");  
messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
```

10. Add the following code after the `//French Language` comment:

```
currentLocale= new Locale("fr", "FR");  
messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
```

11. Add the following code after the `//Language Conversion` comment:

```
b1 = new JButton(messages.getString("PlayGame"));  
b2 = new JButton(messages.getString("ViewInstructions"));  
b3 = new JButton(messages.getString("Exit"));  
f.setTitle(messages.getString("Puzzle"));  
JLabel l = new JLabel(messages.getString("Puzzle"));
```


12. Create a **MessagesBundle_en.properties** resource bundle outside the package but within the application and type the following data:

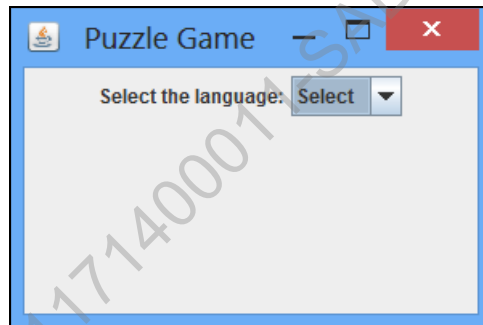
```
PlayGame Play Game
ViewInstructions View Instructions
Exit Exit
Puzzle Puzzle
```

13. Create a **MessagesBundle_fr.properties** resource bundle outside the package but within the application and type the following data:

```
Puzzle Réflexion
PlayGame jouer le jeu
ViewInstructions voir directives
Exit quitter
Puzzle Réflexion
```

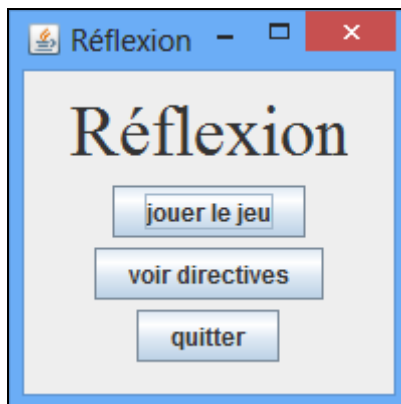
14. Compile and execute the **LocalizationApp** application.

After executing the application, the following output is displayed:



The Output of the LocalizationApp Application

Once you select the language as **French**, the following output is displayed:



The Output of the LocalizationApp Application

Exercises

Exercise 1

Write a program that accepts a Web portal name from the user. In addition, the program should validate the format of the Web portal name. The Web portal name should have the following fields:

- Protocol, such as https or http.
- Domain, such as xyz.
- The top level domain should be more than one character, such as .co.

Exercise 2

Sam needs to write a program to localize the date according to the locale, such as France and US. This program should provide the following functionalities:

- A menu should be displayed to the user that allows the selection of the locale.
- On the basis of the locale selected by the user, the date should be localized and displayed to the user.



Note

*You need to create the classes of the exercises, 1 and 2, inside the package, **chapter03**, inside the application, **Exercises**.*



NIIT

R201171400011-SABHYA

Working with Generics

CHAPTER 4



Activity 4.1: Working with Generics

Problem Statement

Futuristic Inc. is a leading software development organization based in the US. After analysing the communication system of the organization, the management has decided to develop a messaging application for the internal communication. For this, the management has asked Sam, the Senior Software Developer, to develop the application. The management wants the application code to be simple. Sam decides to implement the following functionalities to develop the application:

- The application should display a menu. Using the menu, the user should be able to select either of the two options, SMS or email.
- On the selection of the SMS option, the user should be asked to enter the cell phone number and message.
- On the selection of the email option, the user should be asked to enter the email id and message.
- After entering the required details, the sent message should be processed.

In the initial phase, Sam needs to design the application to implement a handling mechanism that provides the SMS and email services. In addition, to initially test this functionality, he needs to process the sent message by appending the date and displaying the same. Help Sam to achieve the preceding requirements.

Prerequisite

To perform the activity, you need to use the **MessageProcessor.txt** and **MainClass.txt** files.

Solution

To achieve the preceding requirements, Sam needs to perform the following steps:

1. Create a Java application, **MessagingApplication**.
2. Create a package, **messenger**, in the Java application, **MessagingApplication**.
3. Create a class, **EmailMessage**, in the package, **messenger**.
4. Replace the code in the **EmailMessage.java** file with the following code:

```
package messenger;

public class EmailMessage
{
    public String receiver, message;

    public EmailMessage(String receiver, String message)
    {
        this.receiver = receiver;
        this.message = message;
    }
}
```

5. Create a class, **SMSMessage**, in the package, **messenger**.
6. Replace the code in the **SMSMessage.java** file with the following code:

```
package messenger;

public class SMSMessage
{
    public String receiver, message;

    public SMSMessage(String receiver, String message)
    {
        this.receiver = receiver;
        this.message = message;
    }
}
```

7. Create a class, **MessageProcessor**, in the package, **messenger**.
8. Open the **MessageProcessor.txt** file.
9. Replace the code in the **MessageProcessor.java** file with the code in the **MessageProcessor.txt** file.
10. Add the following code snippet after the comment, `//Import Package`, in the **MessageProcessor.java** file:

```
import java.lang.reflect.Field;
```

11. Replace the comment, `//Type Parameter`, in the **MessageProcessor.java** with the following code snippet:

```
<X>
```

12. Add the following code snippet after the comment, `//Generic Variable`, in the **MessageProcessor.java** file:

```
private X message;
```

13. Add the following code snippet after the comment, `//Generic Constructor`, in the **MessageProcessor.java** file:

```
public MessageProcessor(X a1 )
{
    message = a1;
}
```

14. Add the following code snippet after the comment, `//Processing Logic`, in the **MessageProcessor.java** file:

```
public void printResult()
{
    try
    {
        Field receiverField =
message.getClass().getDeclaredField("receiver");

        Object receiverValue = receiverField.get(message);

        Field messagefield = message.getClass().getDeclaredField("message");

        Object messageValue = messagefield.get(message);

        System.out.println("=====");
        System.out.println("Message: " + messageValue + " sent to " +
receiverValue + " was submitted for processing at " + new
java.util.Date().toString());
    }
    catch (Exception ex)
    {
        ex.printStackTrace();
    }
}
```



Note

The `Field` class and the `getDeclaredField()` method of the `Class` class are the part of Reflection API that are used to get the information of a field of a class. The reflected field may be a class (static) field or an instance field.

15. Create a class, **MainClass**, in the package, **messenger**.
16. Open the **MainClass.txt** file.
17. Replace the code in the **MainClass.java** file with the code in the **MainClass.txt** file.
18. Add the following code snippet after the comment, `//SMS Method`, in the **MainClass.java** file:

```
public void getSMSDetails()
{
    Scanner sc = new Scanner(System.in);
    String contact, message;

    System.out.println("=====");
    System.out.print("Enter the contact no: ");
    contact = sc.nextLine();
    System.out.print("\nEnter the message: ");
    message = sc.nextLine();
    SMSMessage sObj = new SMSMessage(contact, message);
    MessageProcessor<SMSMessage> msgObj = new MessageProcessor<>(sObj);
}
```

```

        message1.printResult();
    }

```

19. Add the following code snippet after the comment, //Email Method, in the **MainClass.java** file:

```

public void getEmailDetails()
{
    Scanner sc = new Scanner(System.in);
    String email, message;

    System.out.println("=====");
    System.out.print("Enter the email id: ");
    email = sc.nextLine();
    System.out.print("\nEnter the message: ");
    message = sc.nextLine();
    EmailMessage eObj = new EmailMessage(email, message);
    MessageProcessor<EmailMessage> message2 = new MessageProcessor<>(eObj);
    message2.printResult();
}

```

20. Compile and execute the **MessagingApplication** Java application. After executing the application, the following output is displayed:

```

-----Menu-----
1. SMS
2. Email
3. Exit

```

Choose the option (1/2/3):

Once you provide the option as, 1, the following output is displayed:

```

=====
Enter the contact no:

```

Once you provide the contact no, such as 8767875698, the following output is displayed:

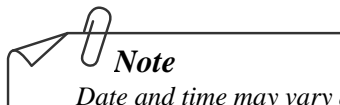
Enter the message:

Once you provide the message, such as Hello, the following output is displayed:

```

=====
Message: Hello sent to 8767875698 was submitted for processing at Sat Jun 15
07:11:46 IST 2013

```



Note

Date and time may vary according to the execution date and time.

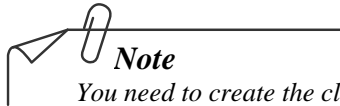
Exercises

Exercise 1

Mark needs to create the ToDoList application. In the initial phase, he has decided to design the application to maintain a track of the list of daily tasks to be completed. For this, Mark has decided to create the following Java classes in the application:

- **Task:** Will store and display the basic details for both the types of tasks, event and meeting.
- **Event:** Will inherit the **Task** class and contain the information of an event.
- **Meeting:** Will inherit the **Task** class and contain the information of a meeting.
- **TaskProcessor:** Will be a generic class that will accept an object of the class that is inherited from the **Task** class. In addition, it will process the tasks, event or meeting.
- **ToDoList:** Will provide a user interface that will allow a user to create an event or meeting by providing the name and time, which is greater than the current time. In addition, it will allow a user to view the created To Do list.

Help Mark to achieve the preceding requirements.



Note

You need to create the class of the exercise, 1, inside the package, **chapter04**, in the application, **Exercises**.

The background of the entire page is a light blue network diagram. It consists of numerous small circular nodes connected by thin, light blue lines, forming a complex web-like structure. The nodes are distributed across the page, with some clusters being denser than others. The overall effect is a subtle, technical, and interconnected aesthetic.

NIIT

R201171400011-SABHYA

Working with Collections

CHAPTER 5



Activity 5.1: Working with Collections

Problem Statement

Turbin Inc. is a leading software development organization based in the US. The organization has got a new client deal to develop the Phone Book application. The management has asked Bob, the Senior Software Developer, to develop the application. The management wants the application code to be simple. In addition, the Phone Book application should be a standalone application. Bob decides to implement the following functionalities in the application:

- The application should display a menu. Using the menu, the user will be able to select either of the two options, add contact or view contact.
- On the selection of the add contact option, the user will be presented with the interface to enter the name and contact number of a person.
- On selecting the Add button, the details must be added to the database. However, Bob has decided to store the contact details in the collections before storing them in the database.
- On the selection of the view contact option, the user will be presented with the list of contacts. The list of contact will be displayed in the form of a list box.
- The user will be able to view the contact details by double-clicking the contact name.

In the initial phase, Bob needs to design the user interface and store the contact details in the collections. Help Bob to achieve the preceding functionalities.

Prerequisite

To perform the activity, you need to use the **Contact.txt**, **AddContact.txt**, and **Menu.txt** files.

Solution

To achieve the preceding functionalities, Bob needs to perform the following steps:

1. Create a Java application, **PhoneBookApplication**.
2. Create a package, **phonebook**, in the Java application, **PhoneBookApplication**.
3. Create a class, **Contact**, in the package, **phonebook**.
4. Open the **Contact.txt** file.
5. Replace the code in the **Contact.java** file with the code in the **Contact.txt** file.
6. Create a class, **AddContact**, in the package, **phonebook**.
7. Open the **AddContact.txt** file.
8. Replace the code in the **AddContact.java** file with the code in the **AddContact.txt** file.
9. Add the following code snippet after the comment, `//Import Packages` in the **AddContact.java** file:

```
import java.util.Comparator;
import java.util.TreeSet;
```

10. Add the following code snippet after the comment, `//Collection Variable` in the **AddContact.java** file:

```
TreeSet<Contact> contactList;
```

11. Add the following code snippet after the comment, `//Sorting Classes` in the **AddContact.java** file:

```
class ContactSort implements Comparator<Contact>
{
    public int compare(Contact c1, Contact c2)
    {
        return c1.getNo().compareTo(c2.getNo());
    }
}

class NameSort implements Comparator<Contact>
{
    public int compare(Contact c1, Contact c2)
    {
        return c1.getName().compareTo(c2.getName());
    }
}
```

12. Add the following code snippet after the comment, `//Collection Initialization` in the **AddContact.java** file:

```
contactList = new TreeSet<Contact>(new ContactSort());
```

13. Add the following code snippet after the comment, `//Action Performed Event` in the **AddContact.java** file:

```
String name = txt_name.getText();
String cno = txt_cno.getText();

if (name.equals("") || (cno.equals("")))
{
    JOptionPane.showMessageDialog(jf_AddContact, "Please fill the details properly", "Warning Message", JOptionPane.WARNING_MESSAGE);
}
else
{
    Contact obj = new Contact();
    obj.setName(name);
    obj.setNo(cno);

    contactList.add(obj);

    ArrayList<Contact> tempList = new ArrayList<Contact>(contactList);
    Collections.sort(tempList, new NameSort());

    dlm_contact.removeAllElements();
}
```

```

        for (Contact c : tempList)
        {
            dlm_contact.addElement(c);
        }

        txt_name.setText("");
        txt_cno.setText("");

JOptionPane.showMessageDialog(jf_AddContact, "Contact has been added",
"Success Message", JOptionPane.INFORMATION_MESSAGE);
int status = JOptionPane.showConfirmDialog(jf_AddContact, "Do you want to add
more contacts?", "Confirmation", JOptionPane.YES_NO_OPTION,
JOptionPane.WARNING_MESSAGE);

        if (status == 0)
        {
            jf_AddContact.setVisible(true);
        }
        else
        {
            jf_AddContact.dispose();
        }
    }
}

```

14. Add the following code snippet after the comment, // Mouse Clicked Event in the **AddContact.java** file:

```

if (e.getSource() == jl_contact)
{
    JList l = (JList) e.getSource();
    if (e.getClickCount() == 2)
    {
        Contact o = (Contact) l.getSelectedValue();
        if (o != null)
        {
            String name = o.getName();
            String no = o.getNo();
            String info = "Name is: " + name + "\nContact No is: " + no;
            JOptionPane.showMessageDialog(jf_AddContact, info, "Contact
Details", JOptionPane.INFORMATION_MESSAGE);
        }
    }
}
}

```

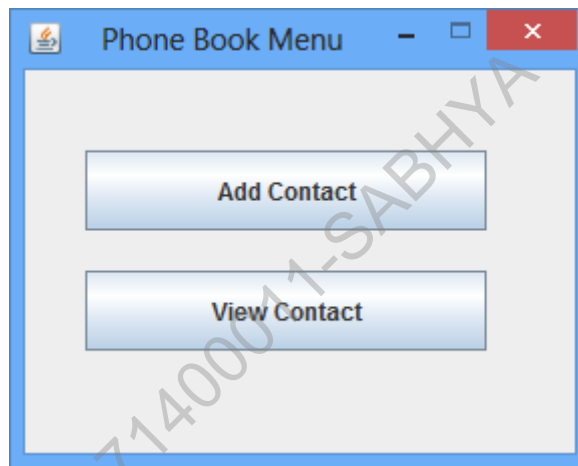
15. Create a class, **Menu**, in the package, **phonebook**.
 16. Open the **Menu.txt** file.
 17. Replace the code in the **Menu.java** file with the code in the **Menu.txt** file.
 18. Add the following code snippet after the comment, //AddContact Object in the **Menu.java** file:

```
AddContact addContObj;
```

19. Add the following code snippet after the comment, `//Action Performed Event` in the **Menu.java** file:

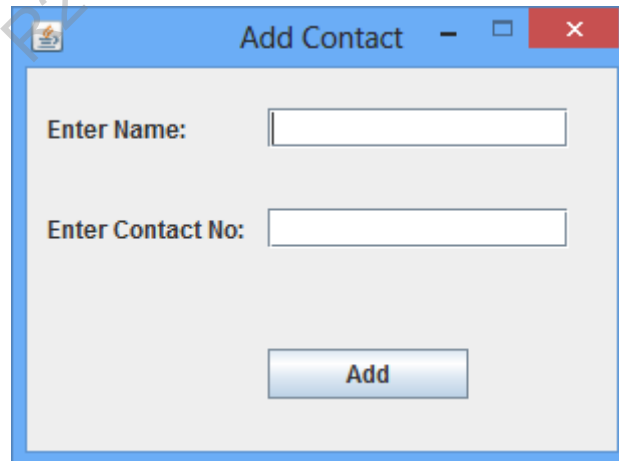
```
if(e.getSource() == btn_add)
{
    addContObj.addComponentToAdd();
}
else if(e.getSource() == btn_view)
{
    addContObj.addComponentToView();
}
```

20. Compile and execute the **PhoneBookApplication** Java application. After executing the application, the following output is displayed:



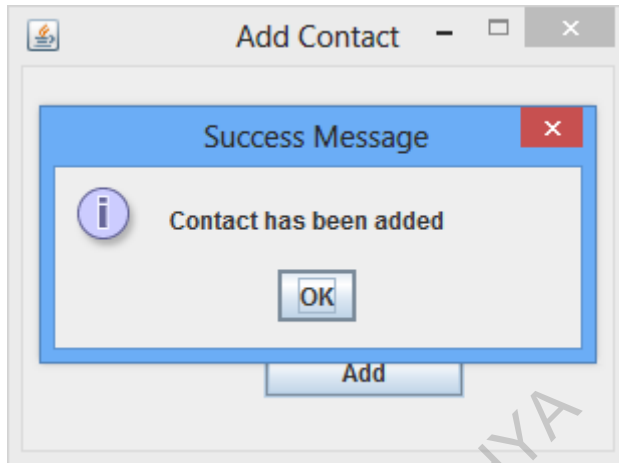
The Output of the PhoneBookApplication Application

Once you click the **Add Contact** button, the following output is displayed:



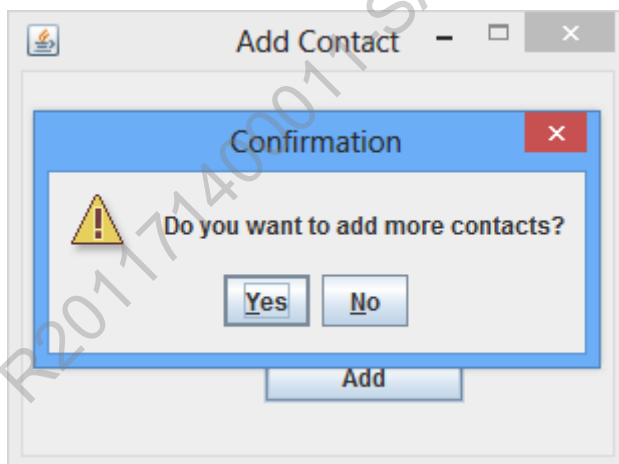
The Output of the PhoneBookApplication Application

Once you enter the name, **Peter Parker**, in the **Enter Name** text box and number, **87483747**, in the **Enter Contact No** text box, and then click the **Add** button, the following output is displayed:



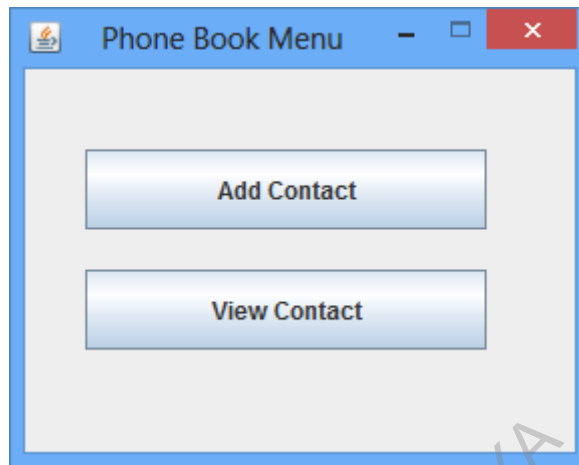
The Output of the PhoneBookApplication Application

Once you click the **OK** button, the following output is displayed:



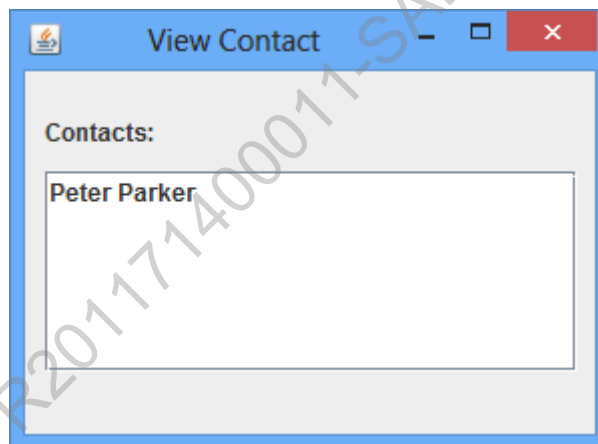
The Output of the PhoneBookApplication Application

Once you click the **No** button, the following output is displayed:



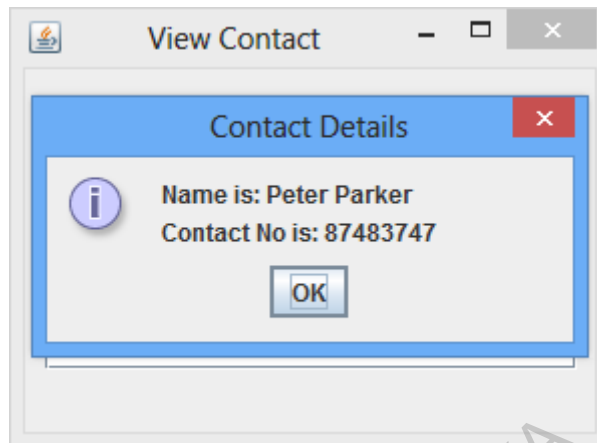
The Output of the PhoneBookApplication Application

Once you click the **View Contact** button, the following output is displayed:



The Output of the PhoneBookApplication Application

Once you double-click **Peter Parker**, the following output is displayed:



The Output of the PhoneBookApplication Application

Click the **OK** button and close the application.

Exercises

Exercise 1

James works as the Software Developer at AxisPro Technologies. He has been assigned the task to create the Fill in the Blanks game for kids. The menu of the game should provide the following options:

1. Play
2. View Instructions
3. Quit

Thereafter, James wants to implement the following functionalities:

- On selecting the Play option:
 - The game should provide a list of categories, such as countries, fruits, and animals.
 - The player will have to create his/her own customized list of one or more categories.
 - The game should randomly select a category from the customized list, and then a word should be chosen randomly from that category and displayed in the form of blanks.
 - The player will have to correctly guess an alphabet for each of the blanks.
 - The game should display an appropriate message when the player correctly guesses all the blanks and wins the game.
- On selecting the View Instructions option, the instructions of the game should be displayed.
- On selecting the Quit option, the game must be terminated.

In the initial phase, James needs to create a game with a list of three categories only. Help James to achieve the preceding functionalities.



Note

You need to create the classes of the exercise, 1, inside the package, **chapter05**, in the application, **Exercises**.

The background of the entire page is a light blue network diagram. It consists of numerous small circular nodes connected by thin, light blue lines, forming a complex web of connections. The nodes are distributed across the page, with some clusters being denser than others. The overall effect is a technical, digital aesthetic.

NIIT

R201171400011-SABHYA

Working with Threads

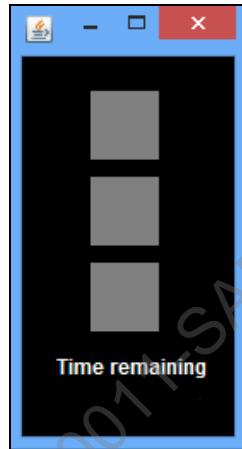
CHAPTER 6



Activity 6.1: Implementing Threads in Java

Problem Statement

Create a GUI-based Java application that simulates the working of the traffic lights by using threads. The GUI should be similar to the interface, as shown in the following figure.



The Traffic Lights GUI

The application must fulfill the following requirements:

1. Initially, when the application is executed, only the RED light must be ON.
2. After 3 seconds, the RED light should turn OFF and only the GREEN light should turn ON for 5 seconds.
3. Then, the GREEN light should turn OFF and only the YELLOW light should turn ON for 2 seconds.
4. Then, the YELLOW light should turn RED.
5. The preceding process should continue, infinitely.
6. You also need to display the remaining time for each light.

Prerequisite

To perform the activity, you need to use the **SignalDemo.txt** file.

Solution

To achieve the preceding requirements, you need to perform the following steps:

1. Create a Java application, **ImplementingThreads**.
2. Create a package, **signal**, in the application, **ImplementingThreads**.
3. Create a class, **SignalDemo**, inside the package, **signal**.
4. Open the **SignalDemo.txt** file.

5. Replace the code in the **SignalDemo.java** file with the code in the **SignalDemo.txt** file.
6. Replace the `//implements runnable` comment with `implements Runnable` in the **SignalDemo.java** file.
7. Add the following code after the `//Add run() method` comment:

```
public void run()
{
    try
    {
        while(true)
        {
            red.setBackground(Color.red);
            yellow.setBackground(Color.GRAY);
            green.setBackground(Color.GRAY);
            for( int i=3;i>0;i--)
            {
                String s = Integer.toString(i);
                show.setText(s+" - Stop - ");
                Thread.sleep(1000);
            }

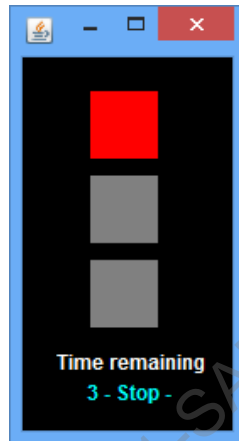
            yellow.setBackground(Color.GRAY);
            green.setBackground(Color.GREEN);
            red.setBackground(Color.GRAY);
            for( int i=5;i>0;i--)
            {
                String s = Integer.toString(i);
                show.setText(s+" - Go - ");
                Thread.sleep(1000);
            }

            yellow.setBackground(Color.YELLOW);
            green.setBackground(Color.GRAY);
            red.setBackground(Color.GRAY);
            for( int i=2;i>0;i--)
            {
                String s = Integer.toString(i);
                show.setText(s+" - Get Slow - ");
                Thread.sleep(1000);
            }
        }
    }
    catch (InterruptedException e)
    {
        System.out.println(e);
    }
}
```

8. Add the following code after the `//create task` comment:

```
Thread task = new Thread(signal);  
task.start();
```

9. Compile and execute the **ImplementingThreads** application.
After executing the application, the following output is displayed:



The Output

Note
The preceding output sequence will continue indefinitely.

Exercises

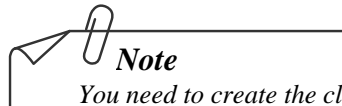
Exercise 1

Create a Java program that creates two threads namely ThreadA and ThreadB. Moreover, you need to ensure that the execution of the threads occurs in such a way that the main method is exited only after both the threads have executed. Furthermore, both the threads should be assigned a sleep time of 5 seconds. The output should be displayed in the following format:

```
Thread A starts
Thread A in for loop, i = 1
Thread A in for loop, i = 2
Thread A in for loop, i = 3
Thread A in for loop, i = 4
Thread A in for loop, i = 5
Thread A in for loop, i = 6
Thread A in for loop, i = 7
Thread A in for loop, i = 8
Thread A in for loop, i = 9
Thread A in for loop, i = 10
Thread A exits
Thread A sleeping for 5 seconds
Thread B starts
Thread B in for loop, i = 1
Thread B in for loop, i = 2
Thread B in for loop, i = 3
Thread B in for loop, i = 4
Thread B in for loop, i = 5
Thread B in for loop, i = 6
Thread B in for loop, i = 7
Thread B in for loop, i = 8
Thread B in for loop, i = 9
Thread B in for loop, i = 10
Thread B exits
Thread B sleeping for 5 seconds
Exit from the main thread
```

Exercise 2

Write a GUI-based application and implement threads to create a notification ticker that will display the message, **!!!Congratulations...!!!**, on the frame. The message should scroll from left to right on the frame. Then, the message should also scroll backwards from right to left. The message scroll must continue infinitely.



Note

*You need to create the classes of the exercises, 1 and 2, inside the package, **chapter06**, in the application, **Exercises**.*

A background network diagram consisting of a grid of light blue squares with a complex web of darker blue lines connecting various nodes, some of which are highlighted with larger blue circles.

NIIT

R201171400011-SABHYA

Implementing Thread Synchronization and Concurrency

CHAPTER 7



Activity 7.1: Implementing Synchronization

Problem Statement

Write an application to simulate the vehicles crossing a toll bridge on a motorway. You need to simulate the environment for five vehicles that are approaching the bridge and tollbooth. The vehicles are numbered from one to five.

The vehicles have to go through the following five stages:

- Start the journey
- Arrive at the toll
- Enter the tollbooth
- Exit the tollbooth
- Cross the bridge

The vehicles can be in any of the stages in parallel, except the third stage, as only one vehicle can enter the tollbooth at a time. The simulation should be performed by creating one thread for each vehicle.

Solution

To achieve the preceding requirements, you need to perform the following steps:

1. Create a Java application, **MotorwaySimulation**.
2. Create a package, **simulation**, in the application, **MotorwaySimulation**.
3. Create a class, **Simulate**, inside the package, **simulation**.
4. Replace the code in **Simulate.java** with the following code:

```
package simulation;
import java.util.Random;

class Vehicle implements Runnable
{
    private int id;    // id identifies each vehicle
    private static TollBooth toll = new TollBooth();    // only one toll booth
    public Vehicle(int id)
    {
        // each vehicle has a different identifying number
        this.id = id;
    }

    public void run()
    {
        System.out.println("Vehicle " + (id+1) + " starts journey");
        Random RandGen = new Random();
        int Rnd = RandGen.nextInt(100);
        travel(Rnd);    // time taken from starting point to toll booth
        System.out.println("Vehicle " + (id+1) + " arrives at the toll");
    }
}
```



```

        toll.useToll(this); // current vehicle uses toll booth
        travel(Rnd); // time taken to cross bridge
        System.out.println("Vehicle " +(id+1)+ " has crossed the bridge");
    }

    public int getVehicleId ()
    {
        return this.id;
    }

    public void travel(int time)
    {
        int limit = 500000;
        for (int j=0; j < time; j++)
        {
            for (int k=0; k < limit; k++)
            {
                // do nothing
            };
        }
    }
}

class TollBooth
{
    boolean inUse;

    public TollBooth()
    {
        inUse = false;
    }

    public void useToll(Vehicle vehicle)
    {
        while (true)
        {
            if (inUse == false)
            {
                synchronized (this)
                {
                    inUse=true;
                    System.out.println("Vehicle
"+(vehicle.getVehicleId()+1)+" enters toolbooth");
                    vehicle.travel(50); // vehicle spends 50 time units in
the toll booth
                    System.out.println("Vehicle
"+(vehicle.getVehicleId()+1)+" exits toolbooth");
                    inUse = false;
                    break;
                }
            }
        }
    }
}
}

```

```

public class Simulate
{
    private static int noOfVehicles = 5;
    private static Vehicle[] vehicles;
    public static void main(String[] args)
    {
        try
        {
            Simulate sm=new Simulate();
            vehicles=new Vehicle[5];
            for(int i=0;i<noOfVehicles;i++)
            {
                vehicles[i]=new Vehicle(i);
                Thread t = new Thread(vehicles[i]);
                t.start();
                Thread.sleep(10);
            }
        }
        catch (Exception ex)
        {
            System.out.println(ex);
        }
    }
}

```

5. Compile and execute the **MotorwaySimulation** Java application.

After executing the application, the following output is displayed:

```

Vehicle 1 starts journey
Vehicle 1 arrives at the toll
Vehicle 1 enters toolbooth
Vehicle 1 exits toolbooth
Vehicle 1 has crossed the bridge
Vehicle 2 starts journey
Vehicle 2 arrives at the toll
Vehicle 2 enters toolbooth
Vehicle 2 exits toolbooth
Vehicle 2 has crossed the bridge
Vehicle 3 starts journey
Vehicle 3 arrives at the toll
Vehicle 3 enters toolbooth
Vehicle 3 exits toolbooth
Vehicle 3 has crossed the bridge
Vehicle 4 starts journey
Vehicle 4 arrives at the toll
Vehicle 4 enters toolbooth
Vehicle 4 exits toolbooth
Vehicle 4 has crossed the bridge
Vehicle 5 starts journey
Vehicle 5 arrives at the toll
Vehicle 5 enters toolbooth
Vehicle 5 exits toolbooth
Vehicle 5 has crossed the bridge

```



Note

The preceding output can vary as the order of thread execution cannot be determined.

R201171400011-SABHYA

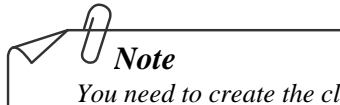
Exercises

Exercise 1

Create a Java program that creates two threads namely Thread1 and Thread2. You need to create a method that prints a table for a specified integer up to 10 numbers. Thread1 should print the table of 2 and Thread2 should print the table of 4. The table values must be printed with a delay of 1 second. You need to ensure that only one thread at a time can print the table. Further, the name of the thread that is printing the table should be displayed.

Exercise 2

Create a Java program that creates two runnable tasks, Display 1 to 10 and Display 11 to 20. The Display 1 to 10 task should print the numbers in the range of 1 to 10 inside a loop. The task, Display 11 to 20, should print the numbers in the range of 11 to 20 inside a loop. Implement the preceding tasks by using an executor service. Ensure that the executor service is shutdown.



Note

You need to create the classes of the exercises, 1 and 2, inside the package, **chapter07**, in the application, **Exercises**.

The background of the entire page is a light blue network diagram. It consists of numerous circular nodes of varying sizes, some of which are highlighted in a slightly darker blue. These nodes are interconnected by a web of thin, light blue lines, creating a complex, organic structure that resembles a molecular or digital network. The overall aesthetic is clean and technical.

NIIT

R201171400011-SABHYA

Working with NIO Classes and Interfaces

CHAPTER 8



Activity 8.1: Performing File Operations Using NIO

Problem Statement

Steve is a programmer in Global Systems Inc. He has been assigned a task to create a Java application that stores the following details about the computer products in a text file:

- Product ID
- Product name
- Product price

In the preceding list, Product ID and Product name should accept a string value. Product price should accept an integer value.

In addition, the application should store the following details about the shops that sell a product:

- Shop ID
- Shop name
- Address

In the preceding list, Shop ID should accept an integer value. Shop name and Address should accept a string value.

As the details are added, they must be appended in the ProductDetails.txt file located in the D:\Details directory. Further, you need to maintain the most recent copy of the preceding file in the D:\Backup directory to ensure a backup.

Prerequisite

You need to create the **Details** and **Backup** folders in the **D:** drive of your computer.

Solution

To achieve the preceding requirements, Steve needs to perform the following steps:

1. Create a Java application, **GlobalSystems**.
2. Create a package, **details**, in the application, **GlobalSystems**.
3. Create a class, **ComputerProducts**, inside the package, **details**.
4. Replace the code in **ComputerProducts.java** with the following code:

```
package details;

import java.io.BufferedWriter;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.*;
import java.util.Scanner;
```

```

public class ComputerProducts {

    String pname, prid, name, address, sid;
    int price;
    String sprid;
    Integer iprice, iid;
    String sprice;
    Path file = Paths.get("D:/Details/ProductDetails.txt");
    Path target = Paths.get("D:/Backup/BackupProductDetails.txt");

    public void createDirectory() throws IOException {
        Path dir = Paths.get("D:/Details");
        Path copydir = Paths.get("D:/Backup");
        enterProductDetails();
    }

    public void enterProductDetails() throws IOException {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the product ID: ");
        prid = input.nextLine();
        System.out.println("Enter the product name: ");
        pname = input.nextLine();
        System.out.println("Enter the product price: ");
        price = input.nextInt();
        sprid = prid;
        iprice = price;
        sprice = iprice.toString();
        printprodDetails();
        enterShopDetails();
    }

    public void enterShopDetails() throws IOException {

        Scanner inputs = new Scanner(System.in);
        System.out.println("Enter the shop ID: ");
        int id = inputs.nextInt();
        System.out.println("Enter the shop name: ");
        name = inputs.next();
        System.out.println("Enter the address: ");
        address = inputs.next();
        iid = id;
        sid = iid.toString();
        System.out.println("Do you want to add more shop details: (Y/N) ");
        Scanner shopchoice = new Scanner(System.in);
        String sch;
        sch = shopchoice.next();

        switch (sch.toUpperCase()) {
            case "Y":
                printshopDetails();
                enterShopDetails();
                break;
            case "N":
                printshopDetails();
        }
    }
}

```

```

        //printDetails();
        System.out.println("Do you want to continue to add product
details: (Y/N) ");
        Scanner choice = new Scanner(System.in);
        String ch;
        ch = choice.next();
        switch (ch.toUpperCase()) {
            case "Y":
                enterProductDetails();

                break;
            case "N":

                System.exit(0);
                break;
            default:
                System.out.println("Invalid entry!");
                break;
        }
        break;
    default:
        System.out.println("Invalid entry!");
        break;
    }
}

public void printprodDetails() throws IOException {
    Charset charset = Charset.forName("US-ASCII");
    try (BufferedWriter writer = Files.newBufferedWriter(file, charset,
StandardOpenOption.CREATE, StandardOpenOption.APPEND)) {
        writer.write("=====");
        writer.newLine();
        writer.append("Product ID: " + sprid + "\n");
        writer.newLine();
        writer.append("Product Name: " + pname + "\n");
        writer.newLine();
        writer.append("Product price: " + sprice + "\n");
        writer.newLine();
        writer.newLine();
    }
}

public void printshopDetails() throws IOException {
    Charset charset = Charset.forName("US-ASCII");
    try (BufferedWriter swriter = Files.newBufferedWriter(file, charset,
StandardOpenOption.CREATE, StandardOpenOption.APPEND)) {
        swriter.newLine();
        swriter.append("Shop ID: " + sid + "\n");
        swriter.newLine();
        swriter.append("Shop name: " + name + "\n");
        swriter.newLine();
        swriter.append("Address: " + address + "\n");
        swriter.newLine();
        swriter.close();
    }
}

```



```

    }

    Path copyfile = Paths.get("D:/Backup/BackupProductDetails.txt");
    if (Files.exists(copyfile)) {
        Files.delete(copyfile);
        Files.copy(file, target);
    } else {
        Files.copy(file, target);
        //enterProductDetails();
    }
}

//// Enter shop details
public static void main(String[] args) throws IOException {
    ComputerProducts obj = new ComputerProducts();
    obj.createDirectory();
}
}

```

5. Compile and execute the **GlobalSystems** Java application.

After executing the application, the following output is displayed:

Enter the product ID:

Once you enter the product id as **M1**, the following output is displayed:

Enter the product name:

Once you enter the product name as **Mouse**, the following output is displayed:

Enter the product price:

Once you enter the product price as **15**, the following output is displayed:

Enter the shop ID:

Once you enter the shop id as **131**, the following output is displayed:


Enter the shop name:

Once you enter the name as **Technosolutions**, the following output is displayed:


Enter the address:

Once you enter the address as **XYZ street**, the following output is displayed:

Do you want to add more shop details: (Y/N)



Note
Further, you can enter Y or N to enter more shop and product details.



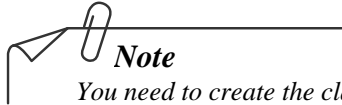
Note
After executing the preceding application, if the application skips any of the inputs, you need to recompile and execute the application.

R201171400011-SABHYA

Exercises

Exercise 1

You need to create a GUI-based file searching application. The application should enable the user to input the name of the file to be searched. Then, on clicking the search button, the file should be searched in the D:\ drive of the computer. Further, the appropriate search results must be displayed to the user.



Note

*You need to create the class of the exercise, 1, inside the package, **chapter08**, in the application, **Exercises**.*

R201171400011-SABHYA

The background of the entire page is a light blue network diagram. It consists of numerous small circular nodes connected by thin, light blue lines, forming a complex web-like structure. The nodes are distributed across the page, with some clusters being denser than others. The overall effect is a technical, digital aesthetic.

NIIT

R201171400011-SABHYA

Introduction to JDBC

CHAPTER 9



Activity 9.1: Creating a JDBC Application to Query a Database

Problem Statement

Create an application to retrieve information (author ID, name, and city) about the authors who are living in the city where the city name begins with the letter, S.

Prerequisite

You need to ensure that the **Library** database exists and comprises the **Authors** table. In addition, the structure of the **Authors** table should be similar to the structure, as shown in the following figure.

Column_name	Type
au_id	varchar
au_name	varchar
phone	varchar
address	varchar
city	varchar
state	varchar
zip	varchar

The Structure of the Authors Table

Solution

To achieve the preceding requirements, you need to perform the following steps:

1. Create a Java application, **Library**.
2. Add the jar file, **sqljdbc4-2.0.jar**, in the **Library** Java application.
3. Create a package, **book**, in the Java application, **Library**.
4. Create a class, **AuthorsInfo**, in the package, **book**.
5. Replace the code in the **AuthorsInfo.java** file with the following code:

```
package book;

import java.sql.*;

public class AuthorsInfo
{
    public static void main(String args[])
    {
        try
        {
            String str = "SELECT * FROM Authors WHERE city LIKE 'S%'";
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

```

        try (Connection con =
DriverManager.getConnection("jdbc:sqlserver://localhost;
databaseName=Library;user=sa;password=password@123");
        Statement stmt = con.createStatement();)
    {
        ResultSet rs = stmt.executeQuery(str);

        System.out.println("Author ID\tAuthor Name\tCity");
        System.out.println("-----");

        while (rs.next())
        {
            String id = rs.getString("au_id");
            String name = rs.getString("au_name");
            String city = rs.getString("city");
            System.out.print(id + "\t");

            if (name.length() <= 7)
            {
                System.out.print(name + "\t\t");
            }
            else
            {
                System.out.print("\t" + name + "\t");
            }
            System.out.println(city);
        }
    }
}
catch (Exception ex)
{
    System.out.println("Error occurred");
    System.out.println("Error:" + ex);
}
}

```

Note

You need to specify the credentials in the preceding code, as per the configuration of SQL Server on your computer.

6. Compile and execute the **Library** Java application. After executing the application, the following output is displayed:

Author ID	Author Name	City
A007	Ringer Albert	Salt Lake City



Note

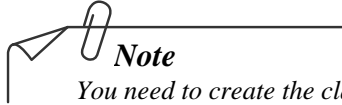
*The preceding output may vary according to the existing values in the **Authors** table of the **Library** database.*

R201171400011-SABHYA

Exercises

Exercise 1

The City library maintenance officer wants to view the names of the books along with their respective author names. Write a program to show the preceding details.



Note

*You need to create the classes of the exercise, 1, inside the package, **chapter09**, in the application, **Exercises**.*

R201171400011-SABHYA

The background of the entire page is a light blue network diagram. It consists of numerous circular nodes of varying sizes, some of which are highlighted in a slightly darker blue. These nodes are interconnected by a web of thin, light blue lines, creating a complex, interconnected pattern that resembles a molecular structure or a data network. The overall aesthetic is clean, modern, and technical.

NIIT

R201171400011-SABHYA

Creating Applications Using Advanced Features of JDBC

CHAPTER 10



Activity 10.1: Creating an Application that Uses the PreparedStatement Object

Problem Statement

The management of City library has decided to computerize the book inventory. The library management has assigned the preceding task to a leading software development company of the US. The Lead Analyst has assigned the development of the Book Inventory application to Mark, the Senior Software Developer. However, in the initial phase, Mark has decided to create an interface that will allow a user to add the details of a new publisher to the publishers table, as shown in the following figure.

PUBLISHERS

PUBLISHERS INFORMATION

Publishers ID:

Publishers Name:

Phone Number:

Address:

City:

State:

Zip:

The User Interface of the Application

Help Mark to achieve the preceding requirement.

Prerequisite

You need to ensure that the **Library** database exists and comprises the **Publishers** table.

The structure of the **Publishers** table should be similar to the structure, as shown in the following figure.

Column_name	Type
pub_id	varchar
pub_name	varchar
phone	varchar
address	varchar
city	varchar
state	varchar
zip	varchar

The Structure of the Publishers Table

Solution

To achieve the preceding requirements, Mark needs to perform the following steps:

1. Create a Java application, **BookInventory**.
2. Add the jar file, **sqljdbc4-2.0.jar**, in the **BookInventory** Java application.
3. Create a package, **inventory**, in the Java application, **BookInventory**.
4. Create a class, **PublisherInfo**, in the package, **inventory**.
5. Replace the code in the **PublisherInfo.java** file with the following code:

```
package inventory;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*.*;
import java.sql.*;

public class PublisherInfo implements ActionListener
{
    JLabel heading, lpubid, lpub_name, lphone, laddress, lcity, lstate, lzip;
    JTextField pub_idField, pub_nameField, phoneField, addressField,
    cityField, stateField, zipField;
    JButton insert, exit;
    Connection con;
    Statement stmt;
    PreparedStatement stat;
    ResultSet rs;
    JPanel p1;
    JFrame f1;

    public PublisherInfo()
    {
        try
        {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

```

        con =
DriverManager.getConnection("jdbc:sqlserver://localhost;databaseName=Library;
user=sa;password=password@123");
        stmt = con.createStatement();

    }
    catch (Exception e)
    {
        System.out.println("Error : " + e);
    }
}

public void compshow()
{
    f1 = new JFrame("PUBLISHERS");
    p1 = new JPanel();
    heading = new JLabel("PUBLISHERS INFORMATION");
    lpubid = new JLabel("Publishers ID:");
    lpub_name = new JLabel("Publishers Name:");
    lphone = new JLabel("Phone Number:");
    laddress = new JLabel("Address:");
    lcity = new JLabel("City:");
    lstate = new JLabel("State:");
    lzip = new JLabel("Zip:");

    pub_idField = new JTextField(6);
    pub_nameField = new JTextField(50);
    phoneField = new JTextField(15);
    addressField = new JTextField(50);
    cityField = new JTextField(50);
    stateField = new JTextField(50);
    zipField = new JTextField(20);

    insert = new JButton("Insert");
    exit = new JButton("Exit");

    p1.setLayout(null);
    heading.setBounds(250, 35, 200, 40);
    p1.add(heading);

    lpubid.setBounds(75, 90, 200, 30);
    pub_idField.setBounds(400, 90, 100, 25);
    p1.add(lpubid);
    p1.add(pub_idField);

    lpub_name.setBounds(75, 120, 200, 30);
    pub_nameField.setBounds(400, 120, 200, 25);
    p1.add(lpub_name);
    p1.add(pub_nameField);

    lphone.setBounds(75, 150, 200, 30);
    phoneField.setBounds(400, 150, 150, 25);
    p1.add(lphone);
    p1.add(phoneField);

```

```

laddress.setBounds(75, 180, 200, 30);
addressField.setBounds(400, 180, 250, 25);
p1.add(laddress);
p1.add(addressField);

lcity.setBounds(75, 210, 200, 30);
cityField.setBounds(400, 210, 200, 25);
p1.add(lcity);
p1.add(cityField);

lstate.setBounds(75, 240, 200, 30);
stateField.setBounds(400, 240, 200, 25);
p1.add(lstate);
p1.add(stateField);

lzip.setBounds(75, 270, 200, 30);
zipField.setBounds(400, 270, 200, 25);
p1.add(lzip);
p1.add(zipField);

insert.setBounds(175, 350, 100, 30);
exit.setBounds(325, 350, 100, 30);
p1.add(insert);
p1.add(exit);

f1.add(p1);
f1.setSize(680, 500);
f1.setVisible(true);
f1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

insert.addActionListener(this);
exit.addActionListener(this);
}

public void actionPerformed(ActionEvent ae)
{
    if (ae.getActionCommand() == "Exit")
    {
        System.exit(0);
    }

    if (ae.getActionCommand() == "Insert")
    {
        if (!pub_idField.getText().equals(""))
        {
            try
            {
                stat = con.prepareStatement("INSERT INTO Publishers VALUES(?,
?, ?, ?, ?, ?)");
                String pid = pub_idField.getText();
                String pname = pub_nameField.getText();
                String pphone = phoneField.getText();
                String padd = addressField.getText();
                String pcity = cityField.getText();

```

```

String pstate = stateField.getText();
String pzip = zipField.getText();

stat.setString(1, pid);
stat.setString(2, pname);
stat.setString(3, pphone);
stat.setString(4, padd);
stat.setString(5, pcity);
stat.setString(6, pstate);
stat.setString(7, pzip);

stat.executeUpdate();

pub_idField.setText("");
pub_nameField.setText("");
phoneField.setText("");
addressField.setText("");
cityField.setText("");
stateField.setText("");
zipField.setText("");
JOptionPane.showMessageDialog(f1, "Information has been
inserted.", "Information", JOptionPane.INFORMATION_MESSAGE);
}
catch (Exception e)
{

    String msg = e.getMessage();

    if(e.getMessage().equals(msg))
    {
        JOptionPane.showMessageDialog(f1, "Record already
exists.", "Warning Message", JOptionPane.WARNING_MESSAGE);
        pub_idField.setText("");
        pub_nameField.setText("");
        phoneField.setText("");
        addressField.setText("");
        cityField.setText("");
        stateField.setText("");
        zipField.setText("");
    }

}
else
{
    JOptionPane.showMessageDialog(f1, "Please fill the details
properly.", "Error Message", JOptionPane.ERROR_MESSAGE);
}

}

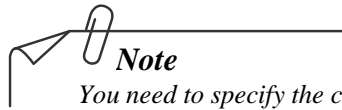
}

```

```

public static void main(String args[])
{
    PublisherInfo p = new PublisherInfo();
    p.compshow();
}

```



Note

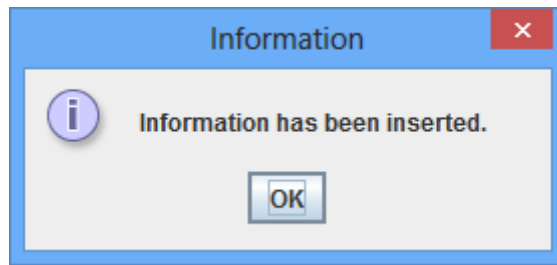
You need to specify the credentials in the preceding code, as per the configuration of SQL Server on your computer.

6. Compile and execute the **BookInventory** Java application. After executing the application, the output is displayed, as shown in the following figure.

The Output of the BookInventory Application

7. Type **P006**, **Park Publishing House**, **0089282389**, **7 Park Street**, **Riverbank**, **California**, and **3338733** in the **Publishers ID**, **Publishers Name**, **Phone Number**, **Address**, **City**, **State**, and **Zip** text boxes, respectively.

8. Click the **Insert** button. The output is displayed, as shown in the following figure.



The Output of the BookInventory Application

9. Click the **OK** button and close the **PUBLISHERS** window.

R201171400011-SABHYA



Activity 10.2: Creating an Application to Determine the Structure of a Table

Problem Statement

The Manager of the New Publishers publishing company often requires the information regarding the tables of the company's database. However, he is not familiar with the SQL statements. Therefore, he has asked Jane, the Software Developer, to create an application that will help him to determine the total number of columns and the data types of the columns of a given table. In addition, Jane needs to ensure that the table name should be specified at runtime.

Prerequisite

You need to ensure that the **Library** database exists and comprises tables, such as **Authors**, **Books**, and **Publishers**.

Solution

To achieve the preceding requirements, Jane needs to perform the following steps:

1. Create a Java application, **NewPublishers**.
2. Add the jar file, **sqljdbc4-2.0.jar**, in the **NewPublishers** Java application.
3. Create a package, **metadata**, in the Java application, **NewPublishers**.
4. Create a class, **ColumnInfo**, in the package, **metadata**.
5. Replace the code in the **ColumnInfo.java** file with the following code:

```
package metadata;

import java.sql.*;
import java.util.Scanner;

public class ColumnInfo
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);

        String tablename;

        try
        {

            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

            try (Connection con =
                DriverManager.getConnection("jdbc:sqlserver://localhost;databaseName=Library;
                user=sa;password=password@123");)
            {
                System.out.println("Enter the table name: ");
```

```

tablename = sc.nextLine();
String str = "SELECT * FROM " + tablename + " ";
try (Statement stmt = con.createStatement();)
{
    ResultSet rs = stmt.executeQuery(str);
    ResultSetMetaData rsmd = rs.getMetaData();
    rs.next();

    System.out.println("=====");
    System.out.println("Number of Attributes in the " +
tablename + " table: " + rsmd.getColumnCount());
    System.out.println("");
    System.out.println("-----");
    System.out.println("Attributes of the " + tablename + "
Table");
    System.out.println("-----");

    for (int i = 1; i <= rsmd.getColumnCount(); i++)
    {
        System.out.println(rsmd.getColumnName(i) + " : " +
rsmd.getColumnTypeName(i));
    }
}
}
catch (Exception e)
{
    System.out.println("Error : " + e);
}
}
}

```

Note

You need to specify the credentials in the preceding code, as per the configuration of SQL Server on your computer.

6. Compile and execute the **NewPublishers** Java application. After executing the application, the following output is displayed:

Enter the table name:

Once you provide the table name as, Publishers, the following output is displayed:

```
=====
Number of Attributes in the Publishers table: 7

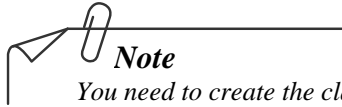
-----
Attributes of the Publishers Table
-----
pub_id : varchar
pub_name : varchar
phone : varchar
address : varchar
city : varchar
state : varchar
zip : varchar
```

R201171400011-SABHYA

Exercises

Exercise 1

The management of Park Library has decided to automate the task of managing the author's details in a database. For this, the library management has assigned the task to a leading software development company of the US. The Senior Software Developer has assigned the development of the Author Management application to Jessica, the Junior Software Developer. For this application, Jessica needs to implement the functionality to view, insert, update, and delete an author's information. Help Jessica to achieve the preceding requirement.



Note

*You need to create the classes of the exercise, 1, inside the package, **chapter10**, in the application, **Exercises**.*

R201171400011-SABHYA

R201171400011-SABHYA

