Java Exception Handling Cheat Sheet

(javaconceptoftheday.com)

Basics

What is exception?

Exception is an abnormal condition which occurs during execution of a program and disrupts the normal flow of a program.

Ex : NumberFormatException, ArithmaticException, ArrayIndexOutOfBoundsException, ClassCastException, NullPointerException, StackOverflowError, OutOfMemoryError etc...

Exception Handling In Java :

Exceptions in Java are handled using try, catch and finally blocks.

```
try
{
   This block contains statements which may
   throw exceptions during run time.
}
catch(Exception e)
{
   This block handles the exceptions thrown by the
   try block.
}
finally
{
   This block is always executed whether an
   exception is thrown or not and thrown exception
   is caught or not.
}
```

Rules To Follow While Writing try-catch-finally Blocks :

- ✓ try, catch and finally blocks form one unit. There must be one try block and one or more catch blocks. finally block is optional.
- ✓ There should not be any statements in between the blocks.
- ✓ If there are multiple catch blocks, the order of catch blocks must be from most specific to general ones. i.e. lower classes in the hierarchy of exceptions must come first and higher classes later.

If try-catch-finally blocks are supposed to return a value :

- ✓ If finally block returns a value then try and catch blocks may or may not return a value.
- ✓ If finally block does not return a value then both try and catch blocks must return a value.
- ✓ finally block overrides return values from try and catch blocks.
- ✓ finally block will be always executed even though try and catch blocks are returning the control.

Frequently Occurring Exceptions

1) NullPointerException occurs when your application tries to access null object.

2) ArrayIndexOutOfBoundsException occurs when you try to access an array element with an invalid index i.e index greater than the array length or with a negative index.

3) NumberFormatException is thrown when you are trying to convert a string to numeric value like integer, float, double etc..., but input string is not a valid number.

4) ClassNotFoundException is thrown when an application tries to load a class at run time but the class with specified name is not found in the classpath.

5) ArithmeticException is thrown when an abnormal arithmetic condition arises in an application.

6) SQLException is thrown when an application encounters with an error while interacting with the database.

7) ClassCastException occurs when an object of one type can not be casted to another type.

8) IOException occurs when an IO operation fails in your application.

9) NoClassDefFoundError is thrown when Java Runtime System tries to load the definition of a class which is no longer available.

10) StackOverflowError is a run time error which occurs when stack overflows. This happens when you keep calling the methods recursively.

Types Of Exceptions

There are two types of exceptions in Java.

1. **Checked Exceptions** are the exceptions which are checked during compilation itself.
2. **Unchecked Exceptions** are the exceptions which are not checked during compilation. They occur only at run time.

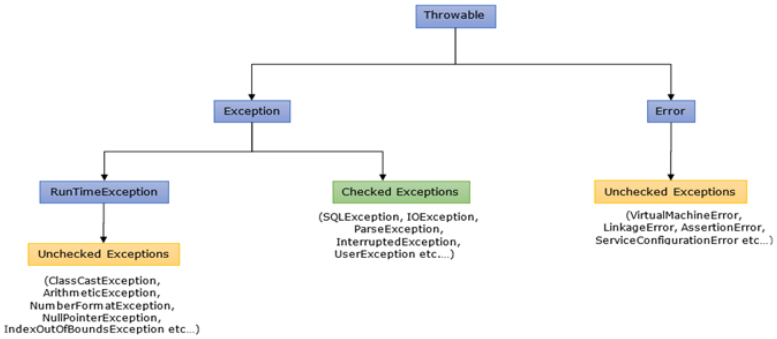
Checked Exceptions	Unchecked Exceptions
They are checked at compile time.	They are not checked at compile time.
They are compile time exceptions.	They are run time exceptions.
These exceptions must be handled properly either using try-catch blocks or using throws clause, otherwise compiler will throw error.	If these exceptions are not handled properly, compiler will not throw any error. But, you may get error at run time.
All the sub classes of java.lang.Exception (except sub classes of java.lang.RunTimeException) are checked exceptions.	All the sub classes of java.lang.RunTimeException and all the sub classes of java.lang.Error are unchecked exceptions.
Ex : FileNotFoundException, IOException, SQLException, ClassNotFoundException	Ex : NullPointerException, ArithmeticException, ClassCastException, ArrayIndexOutOfBoundsException

Hierarchy Of Exceptions

java.lang.Throwable is the super class for all type of errors and exceptions in Java.

It has two sub classes.

1. **java.lang.Error :** It is the super class for all types of errors in Java.
2. **java.lang.Exception :** It is the super class for all types of exceptions in Java.



throw Keyword

throw keyword is used to throw an exception explicitly.

```
try
{
      throw InstanceOfThrowableType;
}
catch(InstanceOfThrowableType)
{

}
```

where, InstanceOfThrowableType must be an object of type Throwable or subclass of Throwable.

throws Keyword

throws keyword is used to specify the exceptions that may be thrown by the method.

```
return_type method_name(parameter_list) throws
exception_list
{
   //some statements
}
```

where, exception_list is the list of exceptions that method may throw. Exceptions must be separated by commas.

Try-with Resources

Try with resources blocks are introduced from Java 7. In these blocks, resources used in try blocks are auto-closed. No need to close the resources explicitly. But, Java 7 try with resources has one drawback. It requires resources to be declared locally within try block. It doesn't recognize resources declared outside the try block. That issue has been resolved in Java 9.

Before Java 7	After Java 7	After Java 9

//Declare resources here
 	
try
{
  //Use resources here
}
catch (Exception e)
{
  //Catch exceptions here if any
}
finally
{
  //Close resources here
}
	try (Declare resources here OR ELSE use local variable referring to a declared resource)
	{
	  //Use resources here
	}
	catch (Exception e)
	{
	  //Catch exceptions here if any
	}
	
	//Resources are auto-closed
	//No need to close resources explicitly
		//Declare resources here
		
		try (Pass reference of declared resources here)
		{
		  //Use resources here
		}
		catch (Exception e)
		{
		  //Catch exceptions here if any
		}
		
		//Resources are auto-closed
		//No need to close resources explicitly

• edureka! •

JAVA OOP CHEAT SHEET

Learn **JAVA** from experts at https://www.edureka.co

Object Oriented Programming in Java

Java is an Object Oriented Programming language that produces software for multiple platforms. An object-based application in Java is concerned with declaring classes, creating objects from them and interacting between these objects.

Java Class

```
class Test {
        // class body
        member variables
        methods
 }
```

Java Object

```
//Declaring and Initializing an object
Test t = new Test();
```

Constructors

Default Constructor

```
class Test{
 /* Added by the Java Compiler at the Run Time
 public Test(){
 }
 */
 public static void main(String args[]) {
   Test testObj = new Test();
  }
}
```

Parameterized Constructor

```
public class Test {
    int appId;
    String appName;
//parameterized constructor with two parameters
Test(int id, String name){
    this.appId = id;
    this.appName = name;
}
void info(){
    System.out.println("Id: "+appId+" Name: "+appName);
}

public static void main(String args[]){
    Test obj1 = new Test(11001,"Facebook");
    Test obj2 = new Test(23003,"Instagram");
    obj1.info();
    obj2.info();
 }
}
```

JAVA CERTIFICATION TRAINING

Inheritance

Single Inheritance

```
Class A {
  //your parent class code
}
Class B extends A {
   //your child class code
}
```

Multi Level Inheritance

```
Class A {
  //your parent class code
}
Class B extends A {
   //your code
}
Class C extends B {
    //your code
}
```

Hybrid Inheritance

```
      A                          A
     / \                         |
    B   C          (OR)         B
   /     \                      / \
  D       E                    C   D
```

Hierarchical Inheritance

```
Class A {
  //your parent class code
}

Class B extends A {
   //your child class code
}

Class C extends A {
    //your child class code
}
```

Multiple Inheritance

```
Class A {
  //your parent class code
}
Class B {
   //your parent class code
}
Class C extends A,B {
    //your child class code
}
```

Polymorphism

Compile Time Polymorphism

```
class Calculator {
  static int add(int a, int b){
  return a+b;
 }
static double add( double a, double b){
  return a+b;
 }
public static void main(String args[]){
  System.out.println(Calculator.add(123,17));
  System.out.println(Calculator.add(18.3,1.9));
 }
}
```

Run Time Polymorphism

```
public class Mobile{
void sms(){System.out.println("Mobile class");}
}

//Extending the Mobile class
public class OnePlus extends Mobile{
//Overriding sms() of Mobile class
 void sms(){
  System.out.println(" OnePlus class");
 }

public static void main(String[] args) {
  OnePlus smsObj= new OnePlus();
  smsObj.sms();
 }
}
```

Abstraction

Abstract Class

```
public abstract class MyAbstractClass
 {
    public abstract void abstractMethod();
    public void display(){
     System.out.println("Concrete method");
  }
 }
```

Interface

```
//Creating an Interface
public interface Bike { public void start(); }
//Creating classes to implement Bike interface
class Honda implements Bike{
  public void start() {
   System.out.println("Honda Bike");
} }
class Apache implements Bike{
  public void start() {
    System.out.println("Apache Bike");
} }
class Rider{
  public static void main(String args[]){
   Bike b1=new Honda();
   b1.start();
   Bike b2=new Apache();
   b2.start();
} }
```

Encapsulation

```
public class Artist {
   private String name;
   //getter method
   public String getName() { return name; }
   //setter method
   public void setName(String name) { this.name = name; }
  }
public class Show{
  public static void main(String[] args){
   //creating instance of the encapsulated class
   Artist s=new Artist();
   //setting value in the name member
   s.setName("BTS");
   //getting value of the name member
   System.out.println(s.getName());
 }
}
```

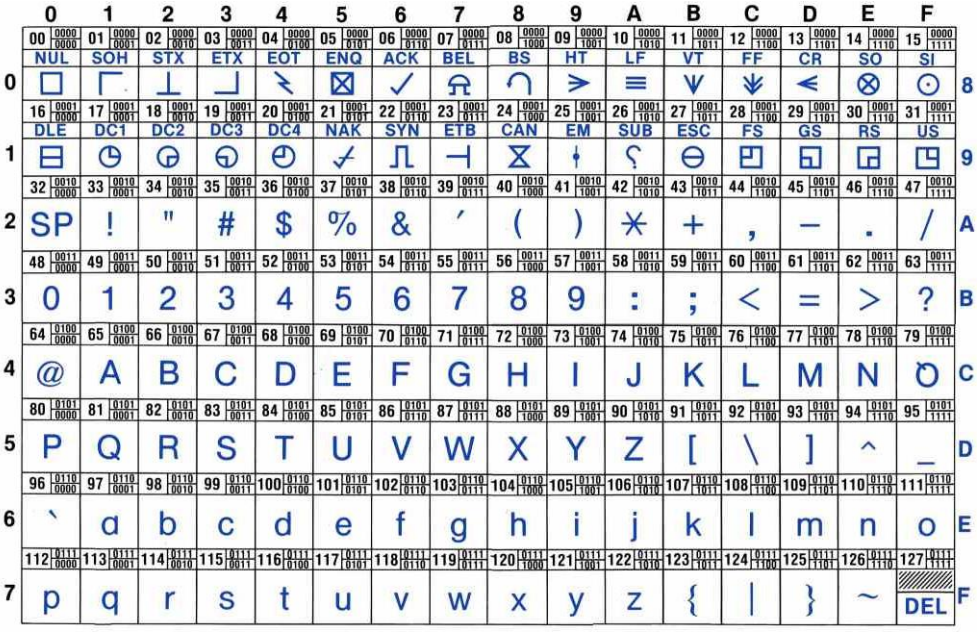
Modifiers in Java

Access Modifiers

Scope	Private	Default	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Non - Access Modifiers

Type	Scope
Static	Makes the attribute dependent on a class
Final	Once defined, doesn't allow any changes
Abstract	Makes the classes and methods abstract
Synchronized	Used to synchronize the threads



```java
package sirikhojornsombut.jedsada.lab6;

// Abstract class Game
public abstract class Game {
    // Instance variables to store game name and number of players
    protected String gameName;
    protected int numOfPlayers;

    // Constructors

    // Default constructor
    public Game() {
        this.gameName = "unknown game";
        this.numOfPlayers = 0;
    }

    // Parameterized constructor
    public Game(String gameName, int numOfPlayers) {
        this.gameName = gameName;
        this.numOfPlayers = numOfPlayers;
    }

    // Getter and setter methods

    // Get the game name
    public String getGameName() {
        return gameName;
    }

    // Set the game name
    public void setGameName(String gameName) {
        this.gameName = gameName;
    }

    // Get the number of players
    public int getNumOfPlayers() {
        return numOfPlayers;
    }

    // Set the number of players
    public void setNumOfPlayers(int numOfPlayers) {
        this.numOfPlayers = numOfPlayers;
    }

    // Abstract method to be implemented by subclasses
    public abstract void playGame();

    // toString() method to provide a string representation of the object

    @Override
    public String toString() {
        return "[gameName=" + gameName + ", numOfPlayers=" + numOfPlayers + "]";
    }
}
```

```java
package sirikhojornsombut.jedsada.lab6;

public interface UseBoard {
    public void setUpBoard();
}
```

```java
package sirikhojornsombut.jedsada.lab5;

import java.util.Scanner;

public class GuessNumberGameVer1 {
    protected int minNum;
    protected int maxNum;
    protected int correctNum;
    protected int maxTries;
    protected static int numOfGames = 0;


    public GuessNumberGameVer1() {
        this.minNum = 1;
        this.maxNum = 10;
        this.correctNum = minNum + (int) (Math.random() * ((maxNum - minNum) + 1));
        this.maxTries = 3;
        numOfGames++;
    }

    // Part 2 Constructors
    public GuessNumberGameVer1(int minNum, int maxNum) {
        this.minNum = minNum;
        this.maxNum = maxNum;
        this.correctNum = minNum + (int) (Math.random() * ((maxNum - minNum) + 1));
        this.maxTries = 3;
        numOfGames++;
    }

    public GuessNumberGameVer1(int minNum, int maxNum, int maxTries) {
        this.minNum = minNum;
        this.maxNum = maxNum;
        this.correctNum = minNum + (int) (Math.random() * ((maxNum - minNum) + 1));
        this.maxTries = maxTries;
        numOfGames++;
    }

    // Play game method
    public void playGame() {
        System.out.println(x:"Welcome to a number guessing game!");
        Scanner scanner = new Scanner(System.in);
        int numberOfTries = 0;



        while (numberOfTries < maxTries) {
            System.out.print("Enter an integer between " + minNum + " and " + maxNum + ":"
            int guess = scanner.nextInt();

            while (guess < minNum || guess > maxNum) {
                System.out.print("Your guess should be in " + minNum + " and " + maxNum +
                guess = scanner.nextInt();
            }

            if (guess == correctNum) {
                System.out.println(x:"Congratulations! You guessed the correct number.");
                break;
            } else {
                if (guess < correctNum) {
                    System.out.println(x:"Try a higher number!");
                } else {
                    System.out.println(x:"Try a lower number!");
```

```java
package sirikhojornsombut.jedsada.lab6;

public class testGame2 {
    Run | Debug
    public static void main(String[] args) {
        GuessNumberGameVer2 game1 = new GuessNumberGameVer2();
        System.out.println(game1);
        GuessNumberGameVer2 game2 = new GuessNumberGameVer2(minNum:1, maxNum:20, maxTries:7);
        System.out.println(game2);
        game2.playGame();
        game2.gameRule();

        MonopolyGameVer2 game3 = new MonopolyGameVer2();
        System.out.println(game3);
        MonopolyGameVer2 game4 = new MonopolyGameVer2(new String[]{"Thimble", "Cat", "Racecar", "Boot"});
        System.out.println(game4);
        game4.playGame();
        game4.setUpBoard();
        game4.gameRule();
        game4.rollDice();

        RockPaperScissorVer2 game5 = new RockPaperScissorVer2();
        System.out.println(game5);
        RockPaperScissorVer2 game6 = new RockPaperScissorVer2(player1Choice:"paper", player2Choice:"rock");
        game6.playGame();
        game6.gameRule();

        Game game7 = new GuessNumberGameVer1();
        System.out.println(game7);
        game7 = new GuessNumberGameVer2();
        System.out.println(game7);
        game7 = new MonopolyGame();
        System.out.println(game7);
        game7 = new MonopolyGameVer2();
        System.out.println(game7);
    }
```

```java
package sirikhojornsombut.jedsada.lab7;

import java.util.Comparator;

public class GuessNumberGameVer4 {
    protected int minNum;
    protected int maxNum;
    protected int maxTries;

    // Constructor for initializing the game parameters
    public GuessNumberGameVer4(int minNum, int maxNum, int maxTries) {
        this.minNum = minNum;
        this.maxNum = maxNum;
        this.maxTries = maxTries;
    }

    // Comparator to sort GuessNumberGameVer4 objects by maxTries
    static class SortByMaxTries implements Comparator<GuessNumberGameVer4> {
        @Override
        public int compare(GuessNumberGameVer4 Guess1, GuessNumberGameVer4 Guess2) {
            return Integer.compare(Guess1.maxTries, Guess2.maxTries);
        }
    }

    // Comparator to sort GuessNumberGameVer4 objects first by maxTries and then by the rar
    static class SortByMaxTriesGuessRange extends SortByMaxTries {

        @Override
        public int compare(GuessNumberGameVer4 Guess1, GuessNumberGameVer4 Guess2) {

            int range1 = Guess1.maxNum - Guess1.minNum;
            int range2 = Guess2.maxNum - Guess2.minNum;

            // Using the super.compare method to compare based on maxTries
            super.compare(Guess1, Guess2);

            // If maxTries comparison is equal, then compare based on the range of numbers
            if (super.compare(Guess1, Guess2) == 0) {
                return Integer.compare(range2, range1);
            }
            // Return the result of maxTries comparison
            return super.compare(Guess1, Guess2);
        }
    }

    // Override toString method to provide a string representation of the object
    @Override
    public String toString() {
        return "GuessNumberGameVer4" + "(min:" + minNum + ", max:" + maxNum + ", max tries:" + maxTries + ")";
    }
}
```

```java
package sirikhojornsombut.jedsada.lab7;

import java.util.ArrayList;
import java.util.Collections;

import sirikhojornsombut.jedsada.lab7.GuessNumberGameVer4.SortByMaxTriesGuessRange

public class TestGamesProb2 {
    static ArrayList<GuessNumberGameVer4> games = new ArrayList<GuessNumberGameVer

    public static void printGamesList(String msg){
        System.out.println(msg);
        for (GuessNumberGameVer4 game : games){
            System.out.println(game);
        }
    }

    public static void intGamesList(){
        games.add(new GuessNumberGameVer4(minNum:1,maxNum:10,maxTries:7));
        games.add(new GuessNumberGameVer4(minNum:1,maxNum:10,maxTries:5));
        games.add(new GuessNumberGameVer4(minNum:1,maxNum:5,maxTries:5));

        printGamesList(msg:"=====Unsorted games list ====");
    }

    public static void sortGamesList(){
        Collections.sort(games, new SortByMaxTriesGuessRange());

        printGamesList(msg:"=====Sorted games list ====");
    }

    Run | Debug
    public static void main(String[] args) {
        intGamesList();
        sortGamesList();
    }
```

```java
import javax.swing.*;
import java.awt.*;

public class MySimpleWindow extends JFrame {
    protected JPanel buttonPanel;
    protected JButton resetButton;
    protected JButton submitButton;

    // Constructor with a title parameter
    public MySimpleWindow(String title) {
        super(title);
    }

    // Method to initialize and add components to the frame
    protected void addComponents() {
        buttonPanel = new JPanel();
        JPanel mainPanel = new JPanel();
        resetButton = new JButton(text:"Reset");
        submitButton = new JButton(text:"Submit");

        buttonPanel.add(resetButton);
        buttonPanel.add(submitButton);

        mainPanel.setLayout(new BorderLayout());
        mainPanel.add(buttonPanel, BorderLayout.SOUTH);

        add(mainPanel);
    }
```

```java
    // Method to set basic frame features
    protected void setFrameFeatures() {
        setLocationRelativeTo(c:null); // Center the frame on the screen
        setVisible(b:true); // Make the frame visible
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Close operation on ex
        pack(); // Pack components within the frame
    }

    // Static method to create and show the GUI
    public static void createAndShowGUI() {
        MySimpleWindow msw = new MySimpleWindow(title:"My Simple Window");
        msw.addComponents();
        msw.setFrameFeatures();
    }

    // Main method to run the application
    Run | Debug
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

```java
public class PlayerFormV1 extends MySimpleWindow {
    protected JPanel topPanel;
    protected JLabel nameLabel;
    protected JLabel nationalityLabel;
    protected JLabel dobLabel;
    protected JTextField nameTextField,nationalityTextField,dobTextField;
    protected JRadioButton maleRadioButton;
    protected JRadioButton femaleRadioButton;

    // Constant for text field length
    public static final int TEXT_FIELD_LENGTH = 15;

    // Constructor for PlayerFormV1
    public PlayerFormV1() {
        super(title:"Player Form V1");
    }

    // Override addComponents to add specific components for PlayerFormV1
    @Override
    protected void addComponents() {
        super.addComponents(); // Call the addComponents method from the su

        topPanel = new JPanel();
        nameLabel = new JLabel(text:"Name:");
        nameTextField = new JTextField(TEXT_FIELD_LENGTH);
        nationalityLabel = new JLabel(text:"Nationality:");
        nationalityTextField = new JTextField(TEXT_FIELD_LENGTH);
        dobLabel = new JLabel(text:"Date of Birth (eg., 31-01-1990):");
        dobTextField = new JTextField(TEXT_FIELD_LENGTH);

        maleRadioButton = new JRadioButton(text:"Male");
        femaleRadioButton = new JRadioButton(text:"Female");
        femaleRadioButton.setSelected(b:true);

        // Create a ButtonGroup for the radio buttons to ensure only one ca
        ButtonGroup genderGroup = new ButtonGroup();
        genderGroup.add(maleRadioButton);
        genderGroup.add(femaleRadioButton);

        topPanel.setLayout(new GridLayout(rows:4, cols:2));
        topPanel.add(nameLabel);
        topPanel.add(nameTextField);
        topPanel.add(nationalityLabel);
        topPanel.add(nationalityTextField);
        topPanel.add(dobLabel);
        topPanel.add(dobTextField);
        topPanel.add(new JLabel(text:"Gender:"));
        JPanel genderPanel = new JPanel();
        genderPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
        genderPanel.add(maleRadioButton);
        genderPanel.add(femaleRadioButton);
        topPanel.add(genderPanel);
```

```java
// Components for favorite sports
protected JList<String> sportsList;
private String[] sports = {"Badminton", "Boxing", "Football", "Running"};

// Constructor for PlayerFormV4
public PlayerFormV4() {
    setTitle(title:"Player Form V4"); // Set the title for PlayerFormV4
}

// Override addComponents to add new components to the frame
protected void addComponents() {
    super.addComponents(); // Call the addComponents method from the parent class

    hobbiesPanel = new JPanel();
    JPanel hobbiesPanel2 = new JPanel();

    hobbiesPanel.setLayout(new GridLayout(rows:0,cols:1));

    // Add checkboxes for hobbies
    JLabel hobbiesLabel = new JLabel(text:"Hobbies:");
    readingCheckbox = new JCheckBox(text:"Reading");
    browsingCheckbox = new JCheckBox(text:"Browsing");
    sleepingCheckbox = new JCheckBox(text:"Sleeping");
    travelingCheckbox = new JCheckBox(text:"Traveling");
    sleepingCheckbox.setSelected(b:true); // Checked by default

    hobbiesPanel.add(hobbiesLabel);

    hobbiesPanel2.setLayout(new FlowLayout(FlowLayout.CENTER));
    hobbiesPanel2.add(readingCheckbox);
    hobbiesPanel2.add(browsingCheckbox);
    hobbiesPanel2.add(sleepingCheckbox);
    hobbiesPanel2.add(travelingCheckbox);

    hobbiesPanel.add(hobbiesPanel2);


    sportPanel = new JPanel();
    sportPanel.setLayout(new GridLayout(rows:0,cols:2));
    // Add components for favorite sports
    JLabel sportLabel = new JLabel(text:"Sport:");
    sportsList = new JList<>(sports);
    sportsList.setSelectedValue(anObject:"Football", shouldScroll:true); // Selected by defaul
    sportPanel.add(sportLabel);
    sportPanel.add(sportsList);

    experiencePanel = new JPanel();
    experiencePanel.setLayout(new GridLayout(rows:0,cols:1));
    // Add components for years of experience slider
    JLabel experienceLabel = new JLabel(text:"Year of experience in this sport:");
    experienceSlider = new JSlider(min:0, max:20, value:0); // Minimum, Maximum, Default
    experienceSlider.setMajorTickSpacing(n:5);
    experienceSlider.setMinorTickSpacing(n:1);
    experienceSlider.setPaintTicks(b:true);
    experienceSlider.setPaintLabels(b:true);
    experiencePanel.add(experienceLabel);
    experiencePanel.add(experienceSlider);
```

```java
public class PlayerFormV2 extends PlayerFormV1 {
    protected JLabel playerTypeLabel;
    protected JComboBox<String> typesCombo;
    protected JLabel noteLabel;
    protected JTextArea noteTextArea;
    protected JPanel centerPanel;

    // Constructor for PlayerFormV2
    public PlayerFormV2() {
        setTitle(title:"Player Form V2"); // Set the title for PlayerFormV2
    }

    // Override addComponents to add specific components for PlayerFormV2
    @Override
    protected void addComponents() {
        super.addComponents(); // Call the addComponents method from the superclass

        playerTypeLabel = new JLabel(text:"Player Type:");
        String[] playerTypes = {"Beginner", "Amateur", "Professional"};
        typesCombo = new JComboBox<>(playerTypes);
        typesCombo.setSelectedItem(anObject:"Amateur");
        typesCombo.setEditable(aFlag:false);

        noteLabel = new JLabel(text:"Note:");
        noteTextArea = new JTextArea(text:"Thailand will face Oman at the Abdullah bin Khalifa Stadium in Doha, Qatar, on Sunday in their second match of the 2023 AFC Asian Cup, Group F.", rows:3, columns:35);
        noteTextArea.setLineWrap(wrap:true);
        noteTextArea.setWrapStyleWord(word:true);
        JScrollPane scrollPane = new JScrollPane(noteTextArea);

        topPanel.setLayout(new GridLayout(rows:5, cols:2));
        topPanel.add(playerTypeLabel);
        topPanel.add(typesCombo);

        centerPanel = new JPanel();
        centerPanel.setLayout(new GridLayout(rows:2, cols:1));
        centerPanel.add(noteLabel);
        centerPanel.add(scrollPane);

        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BorderLayout());
        mainPanel.add(topPanel, BorderLayout.NORTH);
        mainPanel.add(centerPanel ,BorderLayout.CENTER);
        mainPanel.add(buttonPanel, BorderLayout.SOUTH);
        add(mainPanel);
    }
```

```java
JPanel mainPanel = new JPanel();
mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
mainPanel.add(topPanel);
mainPanel.add(hobbiesPanel);
mainPanel.add(sportPanel);
mainPanel.add(experiencePanel);
mainPanel.add(centerPanel);
mainPanel.add(buttonPanel);
add(mainPanel);

// Add components to the content pane
```

```java
public static void createAndShowGUI() {
    PlayerFormV14 playerForm = new PlayerFormV14(); // Create an
    playerForm.addComponents(); // Add components to the frame
    playerForm.addMenus(); // Add menus to the frame
    playerForm.addListeners(); // Add listeners to the frame
    playerForm.setFrameFeatures(); // Set features for the frame
    playerForm.enableKeyboard(); // Enable keyboard shortcuts
}
```

```java
// Override addMenus to add menu bar with specified menus
protected void addMenus() {
    menuBar = new JMenuBar(); // Create a menu bar

    // Menu "File" with four menu items
    JMenu fileMenu = new JMenu(s:"File");
    newMenuItem = new JMenuItem(text:"New");
    openMenuItem = new JMenuItem(text:"Open");
    saveMenuItem = new JMenuItem(text:"Save");
    exitMenuItem = new JMenuItem(text:"Exit");
    fileMenu.add(newMenuItem);
    fileMenu.add(openMenuItem);
    fileMenu.add(saveMenuItem);
    fileMenu.addSeparator();
    fileMenu.add(exitMenuItem);

    // Menu "Config" with two menu items
    JMenu configMenu = new JMenu(s:"Config");

    colorMenu = new JMenu(s:"Color");
    redMenuItem = new JMenuItem(text:"Red");
    greenMenuItem = new JMenuItem(text:"Green");
    blueMenuItem = new JMenuItem(text:"Blue");
    colorMenu.add(redMenuItem);
    colorMenu.add(greenMenuItem);
    colorMenu.add(blueMenuItem);

    JMenuItem sizeMenu = new JMenu(s:"Size");
    size16MenuItem = new JMenuItem(text:"16");
    size20MenuItem = new JMenuItem(text:"20");
    size24MenuItem = new JMenuItem(text:"24");
    sizeMenu.add(size16MenuItem);
    sizeMenu.add(size20MenuItem);
    sizeMenu.add(size24MenuItem);

    configMenu.add(colorMenu);
    configMenu.add(sizeMenu);

    // Add menus to the menu bar
    menuBar.add(fileMenu);
    menuBar.add(configMenu);

    // Set the menu bar for the frame
    setJMenuBar(menuBar);
}
```

```java
JMenuBar menuBar = new JMenuBar(); // Create a menu bar

// Menu "File" with four menu items
JMenu fileMenu = new JMenu(s:"File");
ImageIcon newIcon = new ImageIcon(filename:"sirikhojornsombut/jedsada/lab8/ICON/New-icon.png");
newMenuItem = new JMenuItem(text:"New",newIcon);
ImageIcon openIcon = new ImageIcon(filename:"sirikhojornsombut/jedsada/lab8/ICON/Open-icon.png");
openMenuItem = new JMenuItem(text:"Open",openIcon);
ImageIcon saveIcon = new ImageIcon(filename:"sirikhojornsombut/jedsada/lab8/ICON/Save-icon.png");
saveMenuItem = new JMenuItem(text:"Save",saveIcon);
exitMenuItem = new JMenuItem(text:"Exit");
fileMenu.add(newMenuItem);
fileMenu.add(openMenuItem);
fileMenu.add(saveMenuItem);
fileMenu.add(exitMenuItem);

// Menu "Config" with two menu items
```

```java
public class PlayerFormV6 extends sirikhojornsombut.jedsada.lab8.PlayerFormV5 implements ActionListener {
    protected String gender;
    protected StringBuilder hobbies;
    protected Object srcObject;
```

```java
public class PlayerFormV8 extends sirikhojornsombut.jedsada.lab9.PlayerFormV7 implements ListSelectionListener {

    public PlayerFormV8() {
        super();
        setTitle(title:"Player Form V8");
    }

    @Override
    public void addListeners() {
        super.addListeners();
        // Add ListSelectionListener to the sportsList
        sportsList.addListSelectionListener(this);
    }

    @SuppressWarnings("deprecation")
    @Override
    public void valueChanged(ListSelectionEvent e) {
        if (!e.getValueIsAdjusting()) {
            // Retrieve selected items from the list
            Object[] selectedSports = sportsList.getSelectedValues();

            if (selectedSports.length > 0) {
                // Build the message for the dialog
                StringBuilder message = new StringBuilder(str:"Selected sports are ");
                for (Object sport : selectedSports) {
                    message.append(sport).append(str:", ");
                }
                // Remove the trailing comma and space
                message.setLength(message.length() - 2);

                // Display a dialog with the selected sports
                JOptionPane.showMessageDialog(this, message.toString());
            }
        }
    }
```

```java
public void addListeners(){
    submitButton.addActionListener(this);
    resetButton.addActionListener(this);
    nameTextField.addActionListener(this);
    nationalityTextField.addActionListener(this);
    dobTextField.addActionListener(this);
}

// ActionPerformed method to handle events when buttons or text fields are interacted with
public void actionPerformed(ActionEvent e) {
    srcObject = e.getSource(); // Get the source of the event

    // Determine the selected gender based on radio buttons
    if(maleRadioButton.isSelected() == true){
        gender = "male";
    } else {
        gender = "female";
    }

    // Use StringBuilder to build a string of selected hobbies
    StringBuilder hobbies = new StringBuilder();
    if (readingCheckbox.isSelected()) {
        hobbies.append(readingCheckbox.getText()).append(str:" ");
    }

    if (browsingCheckbox.isSelected()) {
        hobbies.append(browsingCheckbox.getText()).append(str:" ");
    }

    if (sleepingCheckbox.isSelected()) {
        hobbies.append(sleepingCheckbox.getText()).append(str:" ");
    }

    if (travelingCheckbox.isSelected()) {
        hobbies.append(travelingCheckbox.getText()).append(str:" ");
    }

    // Handle different events based on the source object
    if (srcObject == submitButton) {
        JOptionPane.showMessageDialog(this, nameTextField.getText()+" has nationality as "+ nationalityTextField.getText() +" and was born on "+ dobTextField.getText()
        + ", has gender as " + gender +" ,is a "+typesCombo.getSelectedItem() +" player, has hobbies as "+hobbies+" and plays "+sportsList.getSelectedValuesList());
    } else if (srcObject == resetButton){
        nameTextField.setText(t:"");
        nationalityTextField.setText(t:"");
        dobTextField.setText(t:"");
    }
    if (srcObject == nameTextField){
        JOptionPane.showMessageDialog(this,"Name is changed to " + nameTextField.getText());
    } else if (srcObject == nationalityTextField){
        JOptionPane.showMessageDialog(this,"Nationality is changed to " + nationalityTextField.getText());
    }
    else if (srcObject == dobTextField){
        JOptionPane.showMessageDialog(this,"Date of Birth is changed to " + dobTextField.getText());
    }
}
```

```java
public class PlayerFormV7 extends PlayerFormV6 implements ItemListener {

    // Constructor for PlayerFormV7, setting the title
    public PlayerFormV7() {
        setTitle(title:"Player Form V7");
    }

    // Method to add listeners for various components, extending the superclass's listeners
    public void addListeners(){
        super.addListeners(); // Call the superclass method to add its listeners
        maleRadioButton.addItemListener(this);
        femaleRadioButton.addItemListener(this);
        readingCheckbox.addActionListener(this);
        browsingCheckbox.addActionListener(this);
        sleepingCheckbox.addActionListener(this);
        travelingCheckbox.addActionListener(this);
    }

    // actionPerformed method to handle events when buttons or checkboxes are interacted with
    public void actionPerformed(ActionEvent e) {
        super.actionPerformed(e); // Call the superclass method to handle common actions

        // Check if the source object is an instance of JCheckBox
        if(srcObject instanceof JCheckBox){
            JCheckBox checkBox = (JCheckBox) srcObject;

            // Display a message based on whether the checkbox is selected or not
            if(checkBox.isSelected()){
                JOptionPane.showMessageDialog(this, checkBox.getActionCommand()+" is one of the hobbies");
            } else {
                JOptionPane.showMessageDialog(this, checkBox.getActionCommand()+" is no longer one of the hobbies");
            }
        }
    }

    // itemStateChanged method to handle changes in the state of radio buttons
    public void itemStateChanged(ItemEvent e) {
        Object src = e.getItemSelectable();

        // Check if the source object is an instance of JRadioButton
        if (src instanceof JRadioButton) {
            JRadioButton radioBtn = (JRadioButton) src;

            // Display a message when a radio button's state is changed
            if (radioBtn.isSelected())
                JOptionPane.showMessageDialog(this, "Gender is updated to " + radioBtn.getActionCommand());
        }
```

```java
public class PlayerFormV9 extends PlayerFormV8 implements ChangeListener {

    // Constructor for PlayerFormV9
    public PlayerFormV9() {
        super(); // Call the constructor of the superclass (PlayerFormV8)
        setTitle(title:"Player Form V9"); // Set the title for the frame
    }

    // Override method to add listeners
    @Override
    public void addListeners() {
        super.addListeners(); // Call the method from the superclass to add existing listeners
        experienceSlider.addChangeListener(this); // Add ChangeListener to the experienceSlider
    }

    // Override method for stateChanged event in ChangeListener
    @Override
    public void stateChanged(ChangeEvent e) {
        JSlider slider = (JSlider) e.getSource(); // Get the source of the event
        if (!slider.getValueIsAdjusting()) { // Check if the slider value is not adjusting
            int value = slider.getValue(); // Get the current value of the slider
            JOptionPane.showMessageDialog(this, "Year of experience in this sport is " + value); // Display a message dialog with the experience value
        }
    }
```

```java
public void actionPerformed(ActionEvent e) {
    super.actionPerformed(e); // Call the method from the superclass
    srcObject = e.getSource(); // Get the source of the event

    // Check which menu item was clicked and perform corresponding actions
    if (srcObject == newMenuItem) {
        JOptionPane.showMessageDialog(this, message:"You click menu New");
    }
    if (srcObject == openMenuItem) {
        JOptionPane.showMessageDialog(this, message:"You click menu Open");
    }
    if (srcObject == saveMenuItem) {
        JOptionPane.showMessageDialog(this, message:"You click menu Save");
    }
    if (srcObject == exitMenuItem) {
        System.exit(status:0);
    }
    if (srcObject == redMenuItem) {
        // Set text field foreground color to red
        nameTextField.setForeground(Color.RED);
        nationalityTextField.setForeground(Color.RED);
        dobTextField.setForeground(Color.RED);
    }
    if (srcObject == greenMenuItem) {
        // Set text field foreground color to green
        nameTextField.setForeground(Color.GREEN);
        nationalityTextField.setForeground(Color.GREEN);
        dobTextField.setForeground(Color.GREEN);
    }
    if (srcObject == blueMenuItem) {
        // Set text field foreground color to blue
        nameTextField.setForeground(Color.BLUE);
        nationalityTextField.setForeground(Color.BLUE);
        dobTextField.setForeground(Color.BLUE);
    }
    if (srcObject == size16MenuItem) {
        // Set font size of the noteTextArea to 16
        noteTextArea.setFont(new Font(name:"Serif", Font.BOLD, size:16));
    }
    if (srcObject == size20MenuItem) {
        // Set font size of the noteTextArea to 20
        noteTextArea.setFont(new Font(name:"Serif", Font.BOLD, size:20));
    }
    if (srcObject == size24MenuItem) {
        // Set font size of the noteTextArea to 24
        noteTextArea.setFont(new Font(name:"Serif", Font.BOLD, size:24));
    }
}
```

```java
public void actionPerformed(ActionEvent e) {
    super.actionPerformed(e); // Call the method from the superclass
    fileChooser = new JFileChooser();

    // Check if the customMenuItem was clicked
    if (e.getSource() == customMenuItem) {
        // Open JColorChooser dialog to choose text color
        Color newColor = JColorChooser.showDialog(this, title:"Choose Text Color", nameTextField.getForeground());
        if (newColor != null) {
            // Set text field foreground color to the chosen color
            nameTextField.setForeground(newColor);
            nationalityTextField.setForeground(newColor);
            dobTextField.setForeground(newColor);
        }
    }

    // Check if the openMenuItem was clicked
    if (srcObject == openMenuItem) {
        openPlayerDataFromFile();
    }
    // Check if the saveMenuItem was clicked
    if (srcObject == saveMenuItem) {
        savePlayerDataToFile();
    }
}
```

```java
public class Player implements Serializable {
    String name;
    String nationality;
    String dob;
    String playerType;
    String sex;
    ArrayList<String> hobbies;
    ArrayList<String> sports;
    int year;
```

```java
// Enable keyboard shortcuts
public void enableKeyboard() {
    // Set mnemonic keys and accelerator keys
    newMenuItem.setMnemonic(KeyEvent.VK_N);
    openMenuItem.setMnemonic(KeyEvent.VK_O);
    saveMenuItem.setMnemonic(KeyEvent.VK_S);
    exitMenuItem.setMnemonic(KeyEvent.VK_X);

    newMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N, ActionEvent.CTRL_MASK));
    openMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, ActionEvent.CTRL_MASK));
    saveMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S, ActionEvent.CTRL_MASK));
    exitMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X, ActionEvent.CTRL_MASK));
}
```

```java
public void addListeners() {
    super.addListeners();  // Call the addListeners method of the superclass
    nameTextField.setName(name:"Name");  // Set the name for the nameTextField
    nationalityTextField.setName(name:"Nationality");  // Set the name for the nationalityTextField
    dobTextField.setName(name:"Date of Birth");  // Set the name for the dobTextField
}

// Method to handle actions performed (e.g., button clicks)
@Override
public void actionPerformed(ActionEvent e) {
    super.actionPerformed(e);  // Call the actionPerformed method of the superclass
    Object source = e.getSource();  // Get the source of the action event

    // Check which component triggered the action
    if (source == nameTextField) {
        handleTextField(nameTextField, nationalityTextField);
    } else if (source == nationalityTextField) {
        handleTextField(nationalityTextField, dobTextField);
    } else if (source == dobTextField) {
        handleDateTextField(dobTextField);
    }
}

// Method to handle text fields validation
protected void handleTextField(JTextField textField, JTextField nextTextField) {
    String text = textField.getText().trim();  // Get the trimmed text from the text field
    if (text.isEmpty()) {
        // Display a message if the text is empty
        JOptionPane.showMessageDialog(this, "Please enter some data in " + textField.getName());
        textField.requestFocusInWindow();  // Set focus to the current text field
        nextTextField.setEnabled(enabled:false);  // Disable the next text field
    } else {
        nextTextField.setEnabled(enabled:true);  // Enable the next text field if the text is not empty
    }
}
```

```java
// Method to handle date text field validation
protected void handleDateTextField(JTextField textField) {
    String text = textField.getText().trim();  // Get the trimmed text from the date text field
    if (text.isEmpty()) {
        // Display a message if the date is empty
        JOptionPane.showMessageDialog(this, "Please enter a valid date in " + textField.getName());
    } else {
        try {
            // Parse the date using a specified date format
            DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern:"dd-MM-yyyy");
            LocalDate.parse(text, formatter);
            JOptionPane.showMessageDialog(this, textField.getName() + " is changed to " + text);
        } catch (DateTimeParseException e) {
            // Display a message if the date format is invalid
            JOptionPane.showMessageDialog(this, "Please enter a valid date in " + textField.getName());
        }
    }
}


@Override
public void savePlayerDataToFile() {
    int returnVal = fileChooser.showSaveDialog(this);

    if (noButton.isSelected() == true) {
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fileChooser.getSelectedFile();
            try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
                // Write player data in the same format as shown in the dialog
                writer.write(nameTextField.getText() + " has nationality as " + nationalityTextField.getText()
                        + " and was born on " + dobTextField.getText()
                        + ", has gender as " + gender + " ,is a " + typesCombo.getSelectedItem()
                        + " player, has hobbies as " + hobbies + " and plays "
                        + sportsList.getSelectedValuesList());
                JOptionPane.showMessageDialog(this, "Player data saved to " + file.getPath());
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
                } else if (yesButton.isSelected() == true) {
                    File file = fileChooser.getSelectedFile();

                    ArrayList<String> cHobbiesList = new ArrayList<>();
                    if (readingCheckbox.isSelected()) {
                        cHobbiesList.add(e:"readingCheckbox");
                    }

                    if (browsingCheckbox.isSelected()) {
                        cHobbiesList.add(e:"browsingCheckbox");
                    }

                    if (sleepingCheckbox.isSelected()) {
                        cHobbiesList.add(e:"sleepingCheckbox");
                    }

                    if (travelingCheckbox.isSelected()) {
                        cHobbiesList.add(e:"travelingCheckbox");
                    }

                    ArrayList<String> cSportList = new ArrayList<>();
                    cSportList.addAll(sportsList.getSelectedValuesList());

                    Player player = new Player(nameTextField.getText(), nationalityTextField.getText());
                    player.setDob(dobTextField.getText());
                    player.setYear(experienceSlider.getValue());
                    player.setPlayerType((String) typesCombo.getSelectedItem());
                    player.setSex(gender);
                    player.setSports(cSportList);
                    player.setHobbies(cHobbiesList);
                    if (returnVal == JFileChooser.APPROVE_OPTION) {
                        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(file))) {
                            oos.writeObject(player);
                            JOptionPane.showMessageDialog(this, "Saving in file " + file.getPath());
                        } catch (IOException ex) {
                            ex.printStackTrace();
                        }
                    }
                }
```

```java
@Override
public void openPlayerDataFromFile() {
    int returnVal = fileChooser.showOpenDialog(this);

    // Check the selected radio button for saving preferences
    if (noButton.isSelected() == true) {
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fileChooser.getSelectedFile();
            try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
                // Read player data and display in a message dialog
                String data = "";
                String line;
                while ((line = reader.readLine()) != null) {
                    data += line + "\n";
                }
                JOptionPane.showMessageDialog(this, "Opening file " + file.getPath());
                JOptionPane.showMessageDialog(this, "Player data read from file:" + file.getPath() + "\n" + data);
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
} else if (yesButton.isSelected() == true) {
    if (returnVal == JFileChooser.APPROVE_OPTION) {
        File file = fileChooser.getSelectedFile();
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {
            JOptionPane.showMessageDialog(this, "Open file " + file.getPath());
            Player player = (Player) ois.readObject();
            // Fill the form with the attributes of the player object
            if (player.getSex().equals(anObject:"male")) {
                maleRadioButton.setSelected(b:true);
            } else {
                femaleRadioButton.setSelected(b:true);
            }

            nameTextField.setText(player.getName());
            nationalityTextField.setText(player.getNationality());
            dobTextField.setText(player.getDob());
            typesCombo.setSelectedItem((Object) player.getPlayerType());
            experienceSlider.setValue(player.getYear());

            int[] selectedIndices = new int[player.getSports().size()];
            for (int i = 0; i < player.getSports().size(); i++) {
                int index = sportsList.getNextMatch(player.getSports().get(i), startIndex:0, Position.Bias.Forward);
                if (index != -1) {
                    selectedIndices[i] = index;
                }
            }
            sportsList.setSelectedIndices(selectedIndices);

            // Set all checkboxes to false initially
            readingCheckbox.setSelected(b:false);
            browsingCheckbox.setSelected(b:false);
            sleepingCheckbox.setSelected(b:false);
            travelingCheckbox.setSelected(b:false);

            ArrayList<String> aHobbiesList = player.getHobbies();
            for (String hobby : aHobbiesList) {
                switch (hobby) {
                    case "readingCheckbox":
                        readingCheckbox.setSelected(b:true);
                        break;
                    case "browsingCheckbox":
                        browsingCheckbox.setSelected(b:true);
                        break;
                    case "sleepingCheckbox":
                        sleepingCheckbox.setSelected(b:true);
                        break;
                    case "travelingCheckbox":
                        travelingCheckbox.setSelected(b:true);
                        break;
                }
            }
        } catch (IOException | ClassNotFoundException ex) {
            ex.printStackTrace();
        }
    }
}
}
```