Name: Dev M. Bandhiya
Roll No.: SE22UCSE078
Section: CSE1

Skiing Game: (My own version of SkiFree)

The Skiing Game is a Java-based arcade-style game where players control a skier navigating through obstacles like trees, bumps, and "abominators". The game features increasing difficulty through speed progression and tracks the distance traveled.

Imported Libraries and Their Usage

1. **java.awt & javax.swing**:
   - Core libraries for GUI development in Java.
   - **Classes like Color, Graphics, Graphics2D, and Toolkit**: Used for custom rendering.
   - **Image, ImageIcon**: For loading and rendering sprites.
   - **JFrame and JPanel**: For creating the main game window and the game board.
   - **JButton**: For creating interactive buttons like Start, Restart, and Exit.
2. **java.awt.event**:
   - **ActionEvent, ActionListener**: Handling button clicks and timer events.
   - **KeyAdapter, KeyEvent**: Handling keyboard input for skier movement.
3. **javax.swing.Timer**:
   - Manages periodic tasks, such as updating the game state and animations (e.g., snowfall, skier movement).
4. **java.util.Random**:
   - Used for generating random positions and variations (like tree types).
5. **java.util.ArrayList**:
   - For managing dynamic lists of snowflakes for the snowfall animation.

Technologies Used:

1. **Swing:**

   It is part of the Java Foundation Classes (JFC). The main components used from Swing in this game are:

   - **JFrame**: The top-level window containing all the graphical components.
   - **JPanel**: A container used to hold graphical elements and respond to user interactions.
   - **JButton**: Buttons for user interaction (start, restart, and exit buttons).

2. **Graphics2D:**
    - The Graphics2D class, part of Java's 2D API, is used to handle all rendering in the game, from drawing images to text. This class allows for advanced rendering capabilities, such as rotating, scaling, and smoothing images.
3. **Timers:**
    - The Timer class is used to control the timing of events. There are two main timers:
        - One controls the game loop (ActionListener triggers the game's logic every few milliseconds).
        - Another timer updates the distance every second to simulate the passage of time.
4. **KeyEvent & KeyAdapter:**
    - **KeyListener** (via KeyAdapter) is used to capture user input for moving the skier. The game responds to key presses (left, right, up, down) to change the skier's position on the screen.
5. **Random Class:**
    - The **Random** class is used to randomly generate positions for obstacles (trees, bumps, and abominators) on the screen, as well as to generate random images from predefined arrays of images.
6. **ImageIcon:**
    - Images are used for rendering game characters (skier, trees, bumps, and abominators). ImageIcon is used to load images from the file system and display them in the game.

Main Game Mechanisms:

1. Main Game Structure

```
public class Game extends JFrame {
    public Game() {
        add(new Board());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(1024, 768);
        setTitle("Skiing Game");
        setResizable(true);
        setVisible(true);
    }
}
```

The `Game` class serves as the main container, setting up the game window and initializing the game board.

2. Game Board

```
class Board extends JPanel implements ActionListener {
    private Timer timer;
    private Timer distanceTimer;
    private Timer speedTimer;
    private Timer snowflakeTimer;

    private Sk Sk;
    private Tr[] trees;
    private Bp[] bumps;
    private Ab[] abominators;
}
```

The `Board` class manages:

- Game mechanics and state
- Multiple timers for different game aspects
- Game objects (skier, trees, bumps, abominators)
- User input handling
- Collision detection

3. Player Character (Skier)

```
class Sk {
    private Image up1, up2, down, left, right;
    private byte swifter = 0;
    private Image current = left;
    private int dx, dy, x, y;

    public void keyPressed(KeyEvent e) {
        int key = e.getKeyCode();
        if (key == KeyEvent.VK_LEFT) {
            dx = -1;
            current = left;
        }
        // ... other movement controls
    }
}
```

The skier implementation includes:

- Multiple sprite states for different directions
- Movement control through keyboard input
- Position tracking

4. Obstacles System

```
class Tr {
    private Image[] treeImages;
    private Image currentImage;

    public boolean collidesWith(Sk Sk) {
        int skierWidth = Sk.getImage().getWidth(null);
        int skierHeight = Sk.getImage().getHeight(null);
        int treeWidth = currentImage.getWidth(null);
        int treeHeight = currentImage.getHeight(null);

        return x < Sk.getX() + skierWidth && x + treeWidth > Sk.getX() &&
               y < Sk.getY() + skierHeight && y + treeHeight > Sk.getY();
    }
}
```

Trees feature:

- Random selection from multiple sprite variations
- Collision detection with the player
- Vertical scrolling movement

5. Speed Progression

```
speedTimer = new Timer(20000, new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (started && !over) {
            ts += increase;  // Increase speed
            repaint();
        }
    }
});
```

The game implements progressive difficulty through:

- Automatic speed increases every 20 seconds
- Faster obstacle movement
- Distance tracking

6. Snowfall Effect

```
class Sf {
    private int x, y;
    private static final int FALL_SPEED = 2;

    public void moveDown() {
        y += FALL_SPEED;
        if (y > 768) {
            y = 0;
            x = new java.util.Random().nextInt(1024);
        }
    }
}
```

Visual effects include:

- Animated snowfall particles
- Random distribution across the screen
- Continuous recycling of particles

7. Collision System

The game uses a rectangle-based collision detection system:

- Checks for overlaps between player and obstacle hitboxes
- Triggers game over state on collision
- Applies to all obstacle types (trees, bumps, abominators)

OOPs Fundamentals Used:

1. **Inheritance**
    - `Game` class extends `JFrame`
    - `Board` class extends `JPanel`
    - `TAdapter` extends `KeyAdapter`
    - Using inheritance to leverage Java's built-in GUI components and event handling
2. **Encapsulation**
    - Private variables in all classes (x, y, images, etc.)
    - Public methods to access and modify these variables (getX(), getY(), etc.)
    - All game objects encapsulate their state (position, images) and behavior (movement, collision)
    - Example: `Sk` class encapsulates skier's position, images, and movement logic
3. **Abstraction**
    - Complex game mechanics simplified into clear methods
    - Movement logic abstracted in moveDown() methods
    - Collision detection abstracted in collidesWith() methods
    - Timer handling for game updates and animations
4. **Polymorphism**
    - Method Overriding: paint(), actionPerformed(), keyPressed(), keyReleased()
    - Different implementations of moveDown() for different game objects (Tr, Bp, Ab)
5. **Composition**
    - Board contains multiple game objects (Composition over Inheritance)
    - Game contains Board
    - Usage of ArrayLists and arrays to manage multiple objects
6. **Object-Oriented Design Patterns**
    - Event Listener Pattern (ActionListener implementation)
    - Game Loop Pattern (using Timer)
    - Component Pattern (for GUI elements)
7. **Interface Implementation**
    - Board implements ActionListener
    - Usage of event handling interfaces
8. **Data Hiding**
    - Private members with public accessor methods
    - Protected access where needed for inheritance
9. **Class Organization**
    - Each game entity (Sk, Tr, Bp, Ab, Sf) is a separate class
    - Clear separation of concerns
    - Each class has a single responsibility
10. **Message Passing**
    - Objects communicate through method calls
    - Event handling between components
    - Collision detection between objects