Object Oriented Programming

By Watcharin Sarachai

Topic

- Overloading

Overloading

Method Overloading in Java

- Method Overloading is a feature that **allows a class to have more than one method having the same name**, if their argument lists are different.

- It is similar to constructor **overloading** in Java, that allows a class to have more than one constructor having different argument lists.

- When we say argument list it means the parameters that a method has:

  - **For example,** the argument list of a method **add(int a, int b)** having two parameters is different from the argument list of the method **add(int a, int b, int c)** having three parameters.

Three ways to overload a method

- In order to overload a method, the argument lists of the methods must **differ** in either of these:

  1. Number of parameters.

  - For example: This is a valid case of overloading

```
add(int, int)
add(int, int, int)
```

Three ways to overload a method

- In order to overload a method, the argument lists of the methods must **differ** in either of these:

  2. Data type of parameters.

  - For example:

```
add(int, int)
add(int, float)
```

Three ways to overload a method

- In order to overload a method, the argument lists of the methods must **differ** in either of these:

  3. Sequence of Data type of parameters.

  - For example:

```
add(float, int)
add(int, float)
```

Invalid case of method overloading:

- When we say argument list, **we not talking about return type of the method**.
  - **For example,** if two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.

```
int add(int, int)
float add(int, int)
```

Method Overloading examples

- Example 1: Overloading – Different Number of parameters in argument list

```
class DisplayOverloading {
  public void disp(char c) {
    System.out.println(c);
  }

  public void disp(char c, int num) {
    System.out.println(c + " " + num);
  }
}

class Sample {
  public static void main(String args[]) {
    DisplayOverloading obj = new DisplayOverloading();
    obj.disp('a');
    obj.disp('a', 10);
  }
}
```

```
Output:

a
a 10
```

In the above example – method disp() is overloaded based on the number of parameters – We have two methods with the name disp but the parameters they have are different. Both are having different number of parameters.

Method Overloading examples

- Example 2: Overloading – Difference in data type of parameters

```
class DisplayOverloading2 {
  public void disp(char c) {
    System.out.println(c);
  }

  public void disp(int c) {
    System.out.println(c);
  }
}

class Sample2 {
  public static void main(String args[]) {
    DisplayOverloading2 obj = new DisplayOverloading2();
    obj.disp('a');
    obj.disp(5);
  }
}
```

```
Output:

a
5
```

In this example, method disp() is overloaded based on the data type of parameters – We have two methods with the name disp(), one with parameter of char type and another method with the parameter of int type.

Method Overloading examples

- Example3: Overloading – Sequence of data type of arguments

```
class DisplayOverloading3 {
  public void disp(char c, int num) {
    System.out.println("I’m the first definition of method disp");
  }

  public void disp(int num, char c) {
    System.out.println("I’m the second definition of method disp");
  }
}

class Sample3 {
  public static void main(String args[]) {
    DisplayOverloading3 obj = new DisplayOverloading3();
    obj.disp('x', 51);
    obj.disp(52, 'y');
  }
}
```

```
Output:

I’m the first
definition of method
disp
I’m the second
definition of method
disp
```

Method Overloading examples

- Example3: Overloading – Sequence of data type of arguments
  - The previous example method **disp()** is overloaded based on sequence of data type of parameters – Both the methods have different sequence of data type in argument list.
  - First method is having argument list as (char, int) and second is having (int, char). Since the sequence is different, the method can be overloaded without any issues.

Method Overloading and Type Promotion

- When a smaller size of data type is **promoted** to the bigger size, than this is called type **promotion**.
  - **For example:** byte data type can be promoted to short, a short data type can be promoted to int, long, double etc.

What it has to do with method overloading?

- It is very important to understand type promotion. In some case you will think that the program will throw compilation error but in fact that program will run fine because of type promotion.

- Lets take an example to see what we are talking here:

```
class Demo {
  void disp(int a, double b) {
    System.out.println("Method A");
  }

  void disp(int a, double b, double c) {
    System.out.println("Method B");
  }

  public static void main(String args[]) {
    Demo obj = new Demo();
    /*
     * I am passing float value as a second argument but it got promoted to the type
     * double, because there wasn't any method having arg list as (int, float)
     */
    obj.disp(100, 20.67f);
  }
}
```

```
Output:

Method A
```

As you can see that I have passed the float value while calling the **disp()** method but it got promoted to the double type as there wasn't any method with argument list as (int, float)

What it has to do with method overloading?

- This type promotion doesn't always happen:

```java
class Demo {
  void disp(int a, double b) {
    System.out.println("Method A");
  }

  void disp(int a, double b, double c) {
    System.out.println("Method B");
  }

  void disp(int a, float b) {
    System.out.println("Method C");
  }

  public static void main(String args[]) {
    Demo obj = new Demo();
    /*
     * This time promotion won't happen as there is a method with arg list as (int,
     * float)
     */
    obj.disp(100, 20.67f);
  }
}
```

Float

Float

```
Output:

Method C
```

As you see that this time type promotion didn't happen because there was a method with matching argument type.

Type Promotion table:

- The data type on the left side can be promoted to the any of the data type present in the right side of it.

```
byte → short → int → long
short → int → long
int → long → float → double
float → double
long → float → double
```

A few Valid/invalid cases of method overloading

- Case 1:

```
int mymethod(int a, int b, float c)
int mymethod(int var1, int var2, float var3)
```

Result: Compile time error. Argument lists are exactly same. Both methods are having same number, data types and same sequence of data types.

Lets see few Valid/invalid cases of method overloading

- Case 2:

```
int mymethod(int a, int b)
int mymethod(float var1, float var2)
```

Result: Perfectly fine. Valid case of overloading. Here data types of arguments are different.

Lets see few Valid/invalid cases of method overloading

- Case 3:

```
int mymethod(int a, int b)
int mymethod(int num)
```

Result: Perfectly fine. Valid case of overloading. Here number of arguments are different.

Lets see few Valid/invalid cases of method overloading

- Case 4:

```
float mymethod(int a, float b)
float mymethod(float var1, int var2)
```

Result: Perfectly fine. Valid case of overloading. Sequence of the data types of parameters are different, first method is having (int, float) and second is having (float, int).

Lets see few Valid/invalid cases of method overloading

- Case 5:

```
int mymethod(int a, int b)
float mymethod(int var1, int var2)
```

Result: Compile time error. Argument lists are exactly same.
Even though return type of methods are different, it is not a valid case. Since return type of method doesn't matter while overloading a method.

Guess the answers before checking it at the end of programs:

- Question 1 – return type, method name and argument list same.

```
class Demo {
  public int myMethod(int num1, int num2) {
    System.out.println("First myMethod of class Demo");
    return num1 + num2;
  }

  public int myMethod(int var1, int var2) {
    System.out.println("Second myMethod of class Demo");
    return var1 - var2;
  }
}

class Sample4 {
  public static void main(String args[]) {
    Demo obj1 = new Demo();
    obj1.myMethod(10, 10);
    obj1.myMethod(20, 12);
  }
}
```

Guess the answers before checking it at the end of programs:

- Question 1 – return type, method name and argument list same.

Answer:
It will throw a compilation error: More than one method with same name and argument list cannot be defined in a same class.

Guess the answers before checking it at the end of programs:

- Question 2 – return type is different. Method name & argument list same.

```
class Demo2 {
  public double myMethod(int num1, int num2) {
    System.out.println("First myMethod of class Demo");
    return num1 + num2;
  }

  public int myMethod(int var1, int var2) {
    System.out.println("Second myMethod of class Demo");
    return var1 - var2;
  }
}

class Sample5 {
  public static void main(String args[]) {
    Demo2 obj2 = new Demo2();
    obj2.myMethod(10, 10);
    obj2.myMethod(20, 12);
  }
}
```

Guess the answers before checking it at the end of programs:

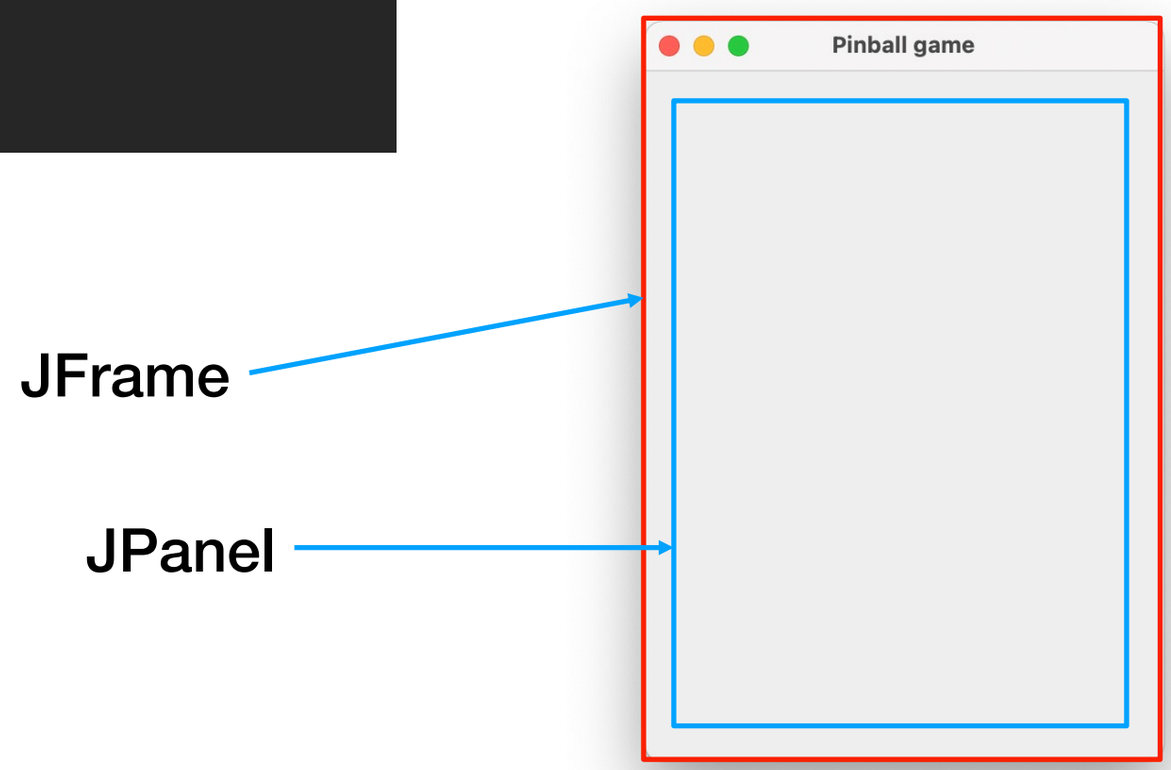
- Question 2 – return type is different. Method name & argument list same.

Answer: It will throw a compilation error: More than one method with same name and argument list cannot be given in a class even though their return type is different. Method return type doesn't matter in case of overloading.

Lab7

- Pinball game

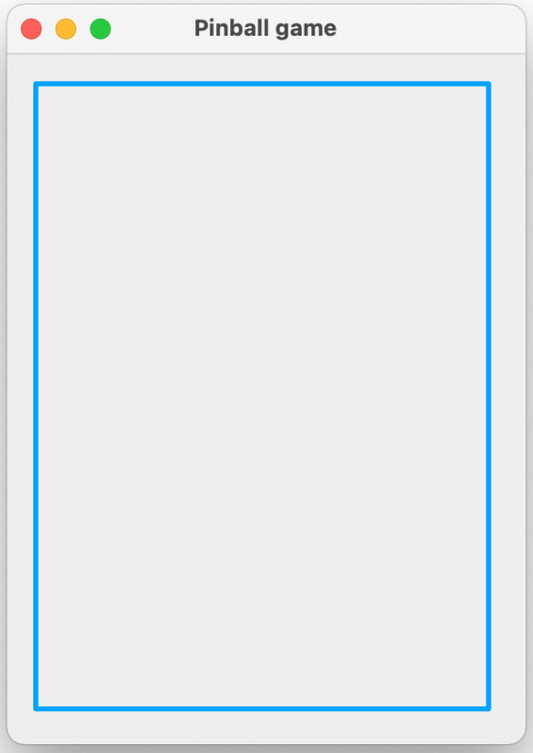
```
JFrame frame = new JFrame("Pinball game");
JPanel panel = new JPanel();
frame.add(panel);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

frame.pack();
frame.setVisible(true);
```



Lab7

- Pinball game
  - Because we need to do more on JPanel, so we must create new class which inherit form JPanel.

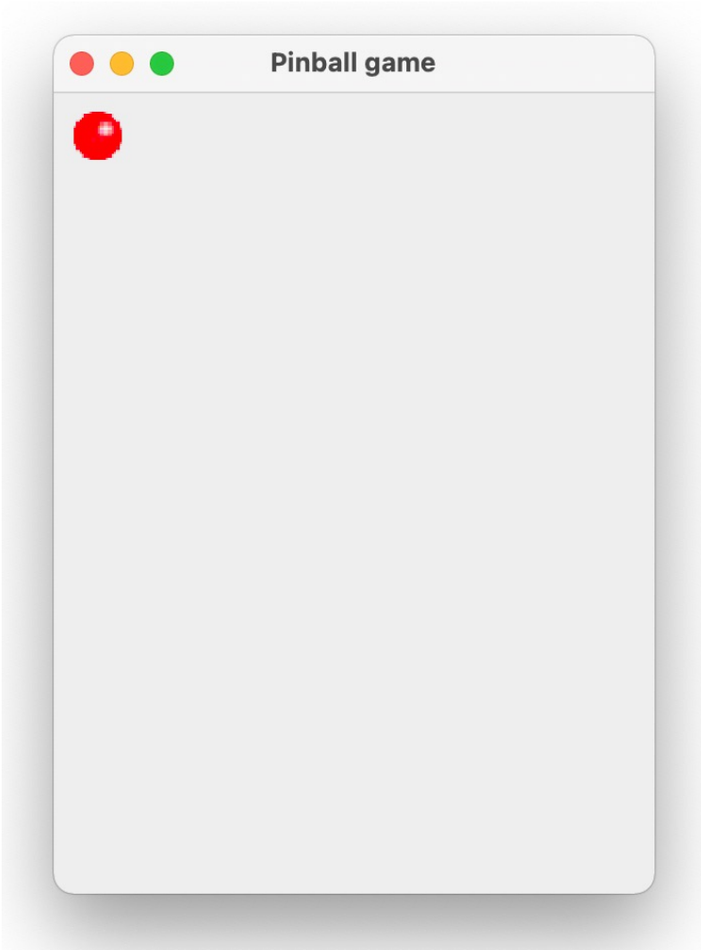
```
public class BackgroundPanel extends JPanel {
  public BackgroundPanel() {
    setPreferredSize(new Dimension(300, 400));
    setBackground(Color.black);
  }

  @Override
  protected void paintComponent(Graphics g) {

  }
}
```



Lab7

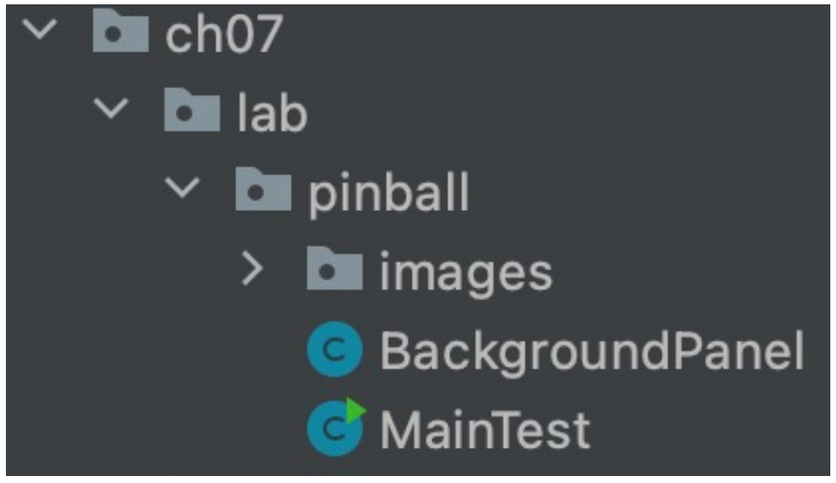
- Pinball game
  - We print some image to JPanel.

```java
public class BackgroundPanel extends JPanel {
  ImageIcon icon;

  BackgroundPanel() {
    setPreferredSize(new Dimension(300, 400));
    setBackground(Color.white);

    URL url = getClass().getResource("images/0.gif");
    icon = new ImageIcon(url);
  }

  @Override
  protected void paintComponent(Graphics g) {
    g.drawImage(icon.getImage(), 10, 10, null);
  }
}
```



```java
public class MainTest {
  public static void main(String [] args) {
    JFrame frame = new JFrame("Pinball game");
    BackgroundPanel panel = new BackgroundPanel();
    frame.add(panel);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
  }
}
```

```java
public class BackgroundPanel extends JPanel {
  ImageIcon icon;

  BackgroundPanel() {
    setPreferredSize(new Dimension(300, 400));
    setBackground(Color.white);

    URL url = getClass().getResource("images/3.gif");
    icon = new ImageIcon(url);
  }

  @Override
  protected void paintComponent(Graphics g) {
    g.drawImage(icon.getImage(), 10, 10, null);
  }
}
```



Lab7

- Pinball game
  - Make the ball bounce left and right; we need a timer that called 32 times per second.
    - Implement the BackgroundPanel with the ActionListener interface. The ActionListener needs the BackgroundPanel to implement the “actionPerformed(ActionEvent e)” method.

```java
public class BackgroundPanel extends JPanel implements ActionListener {
    ImageIcon icon;

    BackgroundPanel() {
        setPreferredSize(new Dimension(300, 400));
        setBackground(Color.white);

        URL url = getClass().getResource("images/0.gif");
        icon = new ImageIcon(url);
    }

    @Override
    protected void paintComponent(Graphics g) {
        g.drawImage(icon.getImage(), 10, 10, null);
    }

    @Override
    public void actionPerformed(ActionEvent e) {

    }
}
```

Lab7

- Pinball game
  - Make the ball bounce left and right; we need a timer that called 32 times per second.
    - In MainTest class. Set 1000/32 to make trigger the panel to call “actionPerformed(ActionEvent e)” 32 times per second.

```
public class MainTest {
    public static void main(String [] args) {
        JFrame frame = new JFrame("Pinball game");
        BackgroundPanel panel = new BackgroundPanel();
        frame.add(panel);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);

        Timer timer = new Timer(1000/32, panel);
        timer.start();
    }
}
```

- In BackgroundPanel class. Edit the code as show below.

```
public class BackgroundPanel extends JPanel implements ActionListener {
    ImageIcon icon;
    private int x;
    private int speedX = 5;

    BackgroundPanel() {
        setPreferredSize(new Dimension(300, 400));
        setBackground(Color.white);

        URL url = getClass().getResource("images/0.gif");
        icon = new ImageIcon(url);
    }

    @Override
    protected void paintComponent(Graphics g) {
        g.drawImage(icon.getImage(), x, 100, null);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        x+=speedX;
        if (x<=0 || x>=getWidth()) {
            speedX = -speedX;
        }
        repaint();
    }
}
```

Lab7

- Pinball game
  - Create new class “Ball” to display a image.
    - Class Ball’s attribute including of:
      - x, y position and speedX, speedY as integer
      - icon image as ImageIcon class
      - Other methods as required to meet the requirement.

- Lab challenge
  - Make a ball bounce around the BackgroundPanel

Reference

- **Super keyword in java with examplehttps://beginnersbook.com/2014/07/super-keyword-in-java-with-example/**
  Accessed: 2020-07-07