Nicholas Liu (nl2523)

Assignment 3 – Theory

1. Two situations where the java compiler can determine the exact method to be called before the system executes is when a method is either static or overloaded.

/\*\*

\* DormRoom class that implements Comparable<DormRoom>. The constructor takes in

\* an occupancy limit andd a floor number

\* @author Nicholas Liu (nl2523)

\*/

public class DormRoom implements Comparable<DormRoom> {

private int occupancyLimit, floorNumber;

/\*\*

\* Constructor for the dorm room

\* @param occupancyLimit Number of people it can hold

\* @param floorNumber Floor number of the dorm room

\*/

public DormRoom(int occupancyLimit, int floorNumber) {

this.occupancyLimit = occupancyLimit;

this.floorNumber = floorNumber;

}

/\*\*

\* Returns the Occupancy Limit of the room

\* @return Returns the occupancyLimit value

\*/

public int getOccupancyLimit() {

return occupancyLimit;

}

/\*\*

\* Returns the floor number

\* @return Returns the floor number of room0

\*/

public int getFloorNumber() {

return floorNumber;

}

/\*\*

\* Returns the higher number floor number

\* @return Returns the room number of the higher floor

\*/

public int compareTo(DormRoom other) {

return Integer.compare(this.floorNumber, other.floorNumber);

}

}

-------------------------------------------------------------------------------------------------------------------------------------

import java.util.ArrayList;

import java.util.Collections;

/\*\*

\* A tester class for the DormRoom object. The dorm rooms are sorted by increasing

\* floor number

\* @author Nicholas Liu (nl2523)

\*/

public class DormRoom\_Tester {

public static void main(String args[]) {

ArrayList<DormRoom> dormRooms = new ArrayList<DormRoom>();

//Fills the dormRooms array list with DormRooms.

for (int x = 0; x < 10; x++) {

dormRooms.add(new DormRoom(x, (int)(Math.random() \* 100)));

}

System.out.println("Before Sorting: ");

for (DormRoom e : dormRooms) {

System.out.print(e.getFloorNumber() + " ");

}

System.out.println("\nAfter Sorting: ");

Collections.sort(dormRooms);

for (DormRoom e : dormRooms) {

System.out.print(e.getFloorNumber() + " ");

}

}

}

3.

/\*\*

\* DormRoom class. The constructor takes in

\* an occupancy limit and a floor number

\* @author Nicholas Liu (nl2523)

\*/

public class DormRoom {

private int occupancyLimit, floorNumber;

/\*\*

\* Constructor for the dorm room

\* @param occupancyLimit Number of people it can hold

\* @param floorNumber Floor number of the dorm room

\*/

public DormRoom(int occupancyLimit, int floorNumber) {

this.occupancyLimit = occupancyLimit;

this.floorNumber = floorNumber;

}

/\*\*

\* Returns the Occupancy Limit of the room

\* @return Returns the occupancyLimit value

\*/

public int getOccupancyLimit() {

return occupancyLimit;

}

/\*\*

\* Returns the floor number

\* @return Returns the floor number of room0

\*/

public int getFloorNumber() {

return floorNumber;

}

}

-------------------------------------------------------------------------------------------------------------------------------------

import java.util.Comparator;

/\*\*

\* DormRoomSortByOccupancy implements the Comparator interface

\*/

public class DormRoomSortByOccupancy implements Comparator<DormRoom> {

public int compare(DormRoom room1, DormRoom room2) {

return Integer.compare(room1.getOccupancyLimit(), room2.getOccupancyLimit());

}

}

-------------------------------------------------------------------------------------------------------------------------------------

import java.util.Comparator;

/\*\*

\* DormRoomSortByFloorNumber implements the Comparator interface

\*/

public class DormRoomSortByFloorNumber implements Comparator<DormRoom> {

public int compare(DormRoom room1, DormRoom room2) {

return Integer.compare(room1.getFloorNumber(), room2.getFloorNumber());

}

}

-------------------------------------------------------------------------------------------------------------------------------------

import java.util.ArrayList;

import java.util.Collections;

/\*\*

\* A tester class for the DormRoom object. The dorm rooms are sorted by increasing

\* floor number

\* @author Nicholas Liu (nl2523)

\*/

public class DormRoom\_Tester {

public static void main(String args[]) {

ArrayList<DormRoom> dormRooms = new ArrayList<DormRoom>();

//Fills the dormRooms array list with DormRooms.

for (int x = 0; x < 10; x++) {

dormRooms.add(new DormRoom((int)(Math.random() \* 8 + 1), (int)(Math.random() \* 100)));

}

System.out.println("Before Sorting: ");

for (DormRoom e : dormRooms) {

System.out.println("Floor Number: " + e.getFloorNumber() + ", Occupancy Limit: " + e.getOccupancyLimit());

}

System.out.println("\nAfter Sorting by Floor Number: ");

Collections.sort(dormRooms, new DormRoomSortByFloorNumber());

for (DormRoom e : dormRooms) {

System.out.println("Floor Number: " + e.getFloorNumber() + ", Occupancy Limit: " + e.getOccupancyLimit());

}

System.out.println("\nAfter Sorting by Occupancy Limit: ");

Collections.sort(dormRooms, new DormRoomSortByOccupancy());

for (DormRoom e : dormRooms) {

System.out.println("Floor Number: " + e.getFloorNumber() + ", Occupancy Limit: " + e.getOccupancyLimit());

}

}

}

4.

Comparator<Country> comp = (country1, country2) -> {return ccountry1.getName().compareTo(country2.getName())};

Assuming this is the first anonymous class implemented in a class called CountrySortTester, the compiler will produce: CountrySortTester$1.class when compiled.

5.

import java.awt.Color;

import java.awt.Component;

import java.awt.FlowLayout;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.geom.Ellipse2D;

import javax.swing.\*;

public class Ass3\_5 implements Icon {

//Defines the Icon size and the default color of the circle

final int ICON\_SIZE = 50;

static Color circleColor = Color.RED;

public static void main(String args[]) {

//Sets up the frame

JFrame frame = new JFrame();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setLayout(new FlowLayout());

//Creates all the buttons and adds it to an array

JButton redButton = new JButton("Red");

JButton greenButton = new JButton("Green");

JButton blueButton = new JButton("Blue");

JButton[] buttons = new JButton[3];

buttons[0] = redButton;

buttons[1] = greenButton;

buttons[2] = blueButton;

//Creates the circle icon and adds to frame

Ass3\_5 myIcon = new Ass3\_5();

JLabel myLabel = new JLabel(myIcon);

frame.add(myLabel);

//Creates an Action Listener for each button

for (JButton b : buttons) {

frame.add(b);

b.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (b.getText().equals("Blue"))

circleColor = Color.BLUE;

else if (b.getText().equals("Green"))

circleColor = Color.GREEN;

else

circleColor = Color.RED;

myLabel.repaint();

}

});

}

//Repacks all the elements

frame.pack();

frame.setVisible(true);

}

/\*\*

\* Returns the ICON\_SIZE variable

\* @return Returns the icon height

\*/

public int getIconHeight() {

return ICON\_SIZE;

}

/\*\*

\* Returns the ICON\_SIZE variable

\* @return Returns the icon height

\*/

public int getIconWidth() {

return ICON\_SIZE;

}

/\*\*

\* Creates a circle with the color of the circleColor variable

\*/

public void paintIcon(Component c, Graphics g, int x, int y) {

Graphics2D g2 = (Graphics2D) g;

Ellipse2D.Double circle = new Ellipse2D.Double(0, 0, 50, 50);

g2.setColor(circleColor);

g2.fill(circle);

}

}