# *Programming II (420-B20-HR)*

# *Lab 15 – Threads*

Date assigned: Wednesday, May 4, 2016

Date due: **Thursday, May 5, 2016**

**Learning Objectives**

At the end of this lab, the student will be able to:

1. write multithreaded programs using a Thread
2. modify the priority of a thread
3. synchronize threads
4. use a MouseListener to respond to mouse events

**To Be Handed In:**

1. The files ***username*\_B20\_L15\_Project** folder should be uploaded to Moodle. Make sure that you have reformatted all your Java classes.

**To Start:**

1. Download and unzip the B20\_L15\_Project folder in your **420-B20\Labs** folder. Rename it to ***username*\_B20\_L15\_Project**.
2. Start **Eclipse**.
3. Create a new Java Project called ***username*\_B20\_L15\_Project**.

# Threads

***Purpose:*** Learn to create a subclass of **Thread** that runs concurrently with a **main()** method.

***To Do:***

## Open the **LetterPrinter** and **Letter** classes in the **threads** package. **LetterPrinter** creates an array of **Letter** objects and assigns a letter of the alphabet to each. It then calls the **print()** method for each **Letter** object to write the letter a specified number of times to a text file. Run the program to see how it works. Open "numbers.txt" to see the order of the output.

## Make a copy of the **Letter** class called **LetterThread**.

## Make the **LetterThread** class a subclass of **Thread**.

## Override the **Thread run()** method in **LetterThread**. It should call **print()**.

## Change the array of **Letter**s in **LetterPrinter** to an array of **LetterThread**s.

## Call the **start()** method instead of calling the **print()** method for each **LetterThread**.

## Run **LetterPrinter** now. Run it several times. How is it different from the previous version using non-threaded objects?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Does it give the same output every time? \_\_\_\_\_\_\_\_

# Priority

***Purpose:*** Learn to give different threads different priorities.

***To Do:***

## Run **LetterPrinter**. In what order do the letters finish?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## We can affect the order in which threads are given access to the CPU using the **setPriority()** method. Add the following statement before starting the threads in **LetterPrinter**:

**letter[i].setPriority(i+1);**

## Modify the **println()** in the **print()** method in the **LetterThread** class to display the **priority** after it displays **ended**. (e.g. If a started first and ended second, the output would be:

**a is finished. Started position: 1. Ended position: 2. Priority: 1.**

## Run **LetterPrinter** again. In what order do the letters finish?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Compare to your answer to the first question. Do you see the effect of the priority?

# Forcing Threads to Sleep

***Purpose:*** Learn to how to force threads to give up the CPU for a given amount of time.

***To Do:***

## You can force a thread to give up the CPU by putting it to sleep for a given time period. In the **run()** method of your **LetterThread** class add

**Thread.sleep(5);**

before calling the **print()** method.

## Compile the class. What happens? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ You must catch an **InterruptedException** when you use the **sleep()** method. Add a **try catch** block to your **run()** method to do this.

## Add a **println()** statement before the call to **sleep()** to print the message "letter *n* is going to sleep." where *n* is the value of the **letter** instance variable.

## Add a **println** statement after the call to **sleep()** to print the message "letter *n* slept for *mm* milliseconds." To calculate *mm,* create two local **long** variables called **startTime** and **endTime**.Use **System.currentTimeMillis()** to initialize **startTime** before calling **sleep()** and **endTime** after calling **sleep()**. *mm* is the difference between the start and end times.

## Run **LetterPrinter** to see the effect of the **sleep()** method. What was the shortest length of time a letter was asleep for? \_\_\_\_\_\_\_\_\_\_\_ Can you see a relationship between time asleep and priority?

# Multithreaded Programs

***Purpose:*** Learn to how to create multiple threads to simultaneously display a graphic.

***To Do:***

## Open the **BouncingBallFrame** class. It is supposed to display a frame with a ball bouncing from the bottom of the frame to the top and back. At the moment, it just displays a ball at the bottom center. Run it to see how it works.

## The **move()** method of the **Ball** class repositions the ball. We want to move the ball each time the screen is drawn. Add **ball.move()** to the end of the **paint()** method.

## Now we want to repaint the frame each time we move the ball. Add **frame.repaint()** to the end of the **move()** method in the **Ball** class.

## Run the frame again. The ball should bounce rapidly up and down.

## Now we want to be able to have several balls of different colours bouncing at the same time. To do this, we will need each ball to be a separate thread. Make the **Ball** class a subclass of **Thread** and override the **run()** method. The **run()** method should call the **move()** method.

## Run the frame again. It should work exactly the same as before.

## Now we want to slow the ball down a bit. We'll use the **sleep()** method to do this. Add a call to **sleep()** for 5 milliseconds after calling **repaint()** in the **move()** method. Correct any errors that occur as a result.

## Run it again. Now we're going to randomly set the colour of the ball.

## Add a **Color** instance variable called **ballColour** to the **Ball** class.

## Add a **static final** array of **Color** objects called **COLOUR** to the **Ball** class. Initialize it using the following:

**{ Color.blue, Color.red, Color.cyan, Color.green, Color.orange, Color.pink, Color.black, Color.magenta }**

## In the **Ball** constructor, generate a random integer between 0 and the length of the **COLOUR** array. Assign **ballColour** to the element of **COLOUR** at the randomized index.

## Add an accessor for **ballColour**.

## In the **BouncingBallFrame** class modify the **g.setColor(Color.blue)** statement in the **paint()** method to use the **ballColour** for the **ball** object.

## Run **BouncingBallFrame** a few times to see if your colour setting is working. (The ball colour should change each time you run it.)

We want to modify the application so that whenever the user clicks the mouse another ball is added to the frame. First, we'll randomize the starting position of the ball along the x-axis.

## In the constructor of the **Ball** class, change the assignment of **xPosition** to:

**xPosition = (int)(Math.random()\* frame.getWidth());**

## Now we want to modify the frame to allow for more than one ball. In the **BouncingBallFrame** class, change the **Ball** object to an array of 100 **Ball** objects. Make all the changes necessary in the class for this to work. At the moment you will only be using **ball[0]**.

## Add an integer instance variable called **numBalls** to keep track of the number of balls in the frame. Initialize it to 1 in the **BouncingBallFrame** constructor.

## Test the application to make sure that it still works.

Now we want to listen for mouse clicks and add a new ball each time the user clicks the mouse:

## Implement **MouseListener** in the **BouncingBallFrame**. Click on the light bulb in the left margin and select **Add unimplemented methods**. What methods were added?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## In the constructor, delete the statement to instantiate **ball[0]** and initialize **numBalls** to **0**.

## In the constructor, add a mouse listener:

**addMouseListener(this);**

## Implement **MouseListener** and select *Add unimplemented methods* from the selections provided in the class header error pop-up. Five methods will be added (mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()).

## Code the **mouseClicked()** method using the following algorithm:

### if **numBalls** is less the maximum number allowed by the array size

#### instantiate the next element of the ball array

#### start the new ball

#### increment the number of balls

## Finally we need to modify the **paint()** method to paint all of the balls. Add a for loop around the **setColor()**, **fillOval()** and **move()** method calls to loop through all the balls in play, set the colour for them, draw them and move them. Modify the ball index to use the loop counter.

## Run your revised **BouncingBallFrame** tosee how your changes worked.

Notice that the balls are jerky. We're going to take the control of calling the move from the **BouncingBallFrame** and put it in the **Ball run()** method.

## Delete the **move()** method call from the **paint()** method.

## Add an infinite while loop around the call to **move()** in the **Ball** **run()** method. (To create an infinite loop, just code **while(true)**.)

## Run the frame now. It should look better.