1.00/1.001
Spring 2012
Final Review

Course Topics Overview

Final

1. Control and scope
2. Classes and objects
3. Arrays, ArrayLists

Quiz 1

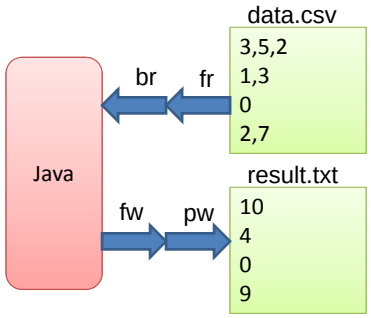
4. Recursion
5. Inheritance
6. Graphical user interfaces

Quiz 2

7. Streams
8. Exceptions
9. Sensors
10. Threads
11. Numerical methods
12. Data structures

7. Streams

1. Different kinds of streams
2. Combining streams
   - FileReader, BufferedReader
   - FileWriter, PrintWriter
3. Reading, writing text
4. Parsing



What does this code do?

```
try{
     FileReader fr = new FileReader("data.csv");
     BufferedReader br = new BufferedReader(fr);

     FileWriter fw = new FileWriter("result.txt");
     PrintWriter pw = new PrintWriter(fw);

     String line;
     while((line = br.readLine())!=null){
          int value = 0;
          String[] svalues = line.split(",");

          for(String svalue: svalues)
               value += Integer.parseInt(svalue);

          pw.println(value);
     }

     br.close();
     pw.close();
}
catch(IOException e){
     System.out.println("IO error occurred.");
}
```

8. Exceptions

1. Syntax
   - try
   - throw
   - catch
2. Inheritance
3. Used with streams

```
public static void main(String[] args) {
    try{
        someMethod( 2);
    }


    catch(FavoriteException e){
        System.out.println("Ack!");
    }
}
```

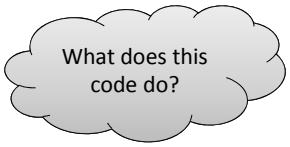
What does this code do?

```
public void someMethod(int a)
            throws FavoriteException
{
    if (a<0)
        throw new FavoriteException();
    System.out.println(a);
}
```

```
class FavoriteException extends Exception {
    public FavoriteException(){ }
}
```

8. Exceptions

1. Syntax
   - try
   - throw
   - catch
2. Inheritance
3. Used with streams



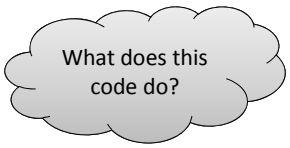

```
public static void main(String[] args) {
    try{
        someMethod(-2);
    }

    catch(FavoriteException e){
        System.out.println("Ack!");
    }
}
```

```
public void someMethod(int a)
            throws FavoriteException
{
    if (a<0)
        throw new FavoriteException();
    System.out.println(a);
}
```

```
class FavoriteException extends Exception {
    public FavoriteException(){ }
}
```

8. Exceptions

1. Syntax
   - try
   - throw
   - catch
2. Inheritance
3. Used with streams




```
public static void main(String[] args) {
    try{
        someMethod(-2);
    }
    catch(Exception e){
        System.out.println("Oops!");
    }
    catch(FavoriteException e){
        System.out.println("Ack!");
    }
}
```

```
public void someMethod(int a)
            throws FavoriteException
{
    if (a<0)
        throw new FavoriteException();
    System.out.println(a);
}
```

```
class FavoriteException extends Exception {
    public FavoriteException(){ }
}
```

9. Sensors

1. Sensor change listener

```
interfaceKit.addSensorChangeListener(new SensorChangeListener() {
        public void sensorChanged(SensorChangeEvent se) {
                if (se.getIndex() == lightIndex)
                        if (se.getValue() < 20) {
                                System.out.println("Light low");
                        }
        }
});
```

2. Compare with action listener

```
button.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent ae){
                System.exit(0);
        }
});
```

10. Threads

1. Process vs. thread
2. Thread class
3. Runnable interface
4. Synchronization

```
public class Parallel extends Thread {

    private static int j = 1;

    public static void main(String[] args) {


        doThis("Main");
        doThis("Parallel");
    }




    public static void doThis(String a){
        for (int i=1; i<=200; i++) {
            System.out.println((j++) + " " + a);




        }
    }
}
```

10. Threads

1. Process vs. thread
2. Thread class
3. Runnable interface
4. Synchronization

Make separate thread

```
public class Parallel extends Thread {

    private static int j = 1;

    public static void main(String[] args) {
        Thread p = new Parallel();
        p.start();
        doThis("Main");

    }

    public void run() {
         doThis("Parallel");
    }

    public static void doThis(String a){
        for (int i=1; i<=200; i++) {
            System.out.println((j++) + " " + a);




        }
    }
}
```

10. Threads

1. Process vs. thread
2. Thread class
3. Runnable interface
4. Synchronization

Make separate thread

Pause 20ms in loop

```
public class Parallel extends Thread {

    private static int j = 1;

    public static void main(String[] args) {
        Thread p = new Parallel();
        p.start();
        doThis("Main");

    }

    public void run() {
         doThis("Parallel");
    }

    public static void doThis(String a){
        for (int i=1; i<=200; i++) {
            System.out.println((j++) + " " + a);
            try {
                Thread.sleep(20);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

10. Threads

1. Process vs. thread
2. Thread class
3. Runnable interface
4. Synchronization

Make separate thread

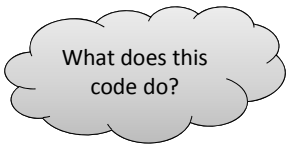
Pause 20ms in loop

Prevent parallel access to j

```
public class Parallel extends Thread {

    private static int j = 1;

    public static void main(String[] args) {
        Thread p = new Parallel();
        p.start();
        doThis("Main");

    }

    public void run() {
         doThis("Parallel");
    }

    public synchronized static void doThis(String a){
        for (int i=1; i<=200; i++) {
            System.out.println((j++) + " " + a);
            try {
                Thread.sleep(20);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

11. Numerical methods

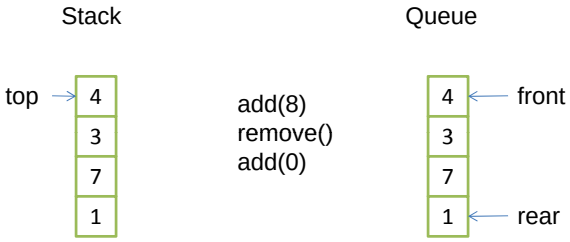
1. Matrices
   - Based on 2D array
   - Matrix manipulation
2. Integration methods
3. Root finding methods




```
public class Matrix{
    private double[][] data;
    // other methods
    public Matrix doSomething(){
        int rows = data.length;
        int cols = data[0].length;
        Matrix mat = new Matrix(rows,cols);
        for (int i=0; i<rows; i++)
            for (int j=0; j<cols; j++)
                mat.setElement(i,j, -data[i][j]);
        return mat;
    }
}
```

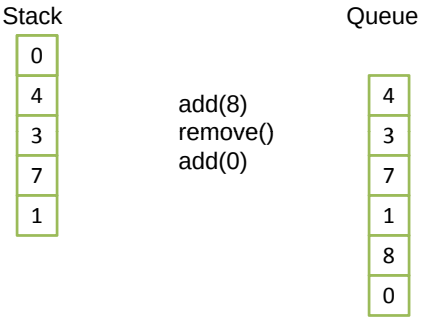
12. Data structures

1. Sorting
   - Comparable
   - Comparator
2. Stacks and queues
3. LinkedList
   - List traversal
4. TreeMap, HashMap



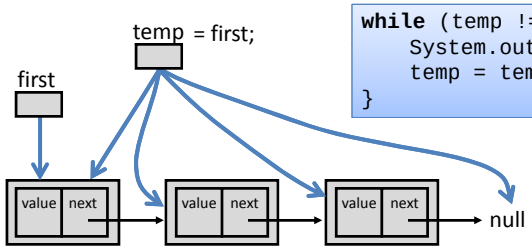
12. Data structures

1. Sorting
   - Comparable
   - Comparator
2. Stacks and queues
3. LinkedList
   - List traversal
4. TreeMap, HashMap



12. Data structures

1. Sorting
   - Comparable
   - Comparator
2. Stacks and queues
3. LinkedList
   - List traversal
4. TreeMap, HashMap




```
while (temp != null) {
    System.out.println(temp.value);
    temp = temp.next;
}
```

Previous Finals

Fall 2011

1. True/false
2. Data structures
3. **Matrices**
4. **Streams**
5. **Sensors**

Spring 2010

1. **Classes, inheritance**
2. **Exceptions**
3. Sorting, hashing
4. **Matrices and recursion**
5. **Streams and Swing**
6. Data structures

1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012