Agenda

- Review\ Q&A last lecture and assignment1
- Quiz (20 minutes)
- Exceptions
- Array
- Array List
- Enhanced for loop

Exceptions

- What is an exception…???

Exception

- It is an unexpected problem that arises during the execution of the program.
- It terminates the program/application abnormally.

Error Vs. Exception

- **Error:-** An Error indicates serious problem that a reasonable application should not try to catch.

- **Exception:-** Exception indicates conditions that a reasonable application might try to catch.

Reasons for exception

- Invalid data entered by user.
- File cannot be found, that needs to be open
- Lost of connection in the middle of communications or the JVM has ran out of memory.

Contd.,

- Exceptions are caused due to:

✓User errors,

✓Programmer error

✓Physical resources that have failed

Categories of exception

There are 3 categories of exception

- Checked Exception
- Unchecked Exception
- Errors

Checked exception

- This type of exception is checked(notified) by the complier at the time of compilation, also called ‘compile time exceptions’.
- These exceptions should be handled by programmer.

For example:

“IllegalArgumentException”

Entered integer where String is expected and vice a versa

Unchecked exceptions

- Unchecked exception occurs at the time of execution. Also known as RunTime Exception.
- These include programming bugs
- These exceptions are ignored at the time of compilation.

For example:

Declaring an array of size 10 and trying to call 11^{th} element

ArrayIndexOutOfBoundsException exception occurs.

Errors

- These are not exception but problems that arise beyond the control of the user or the programmer.
- Errors are typically ignored because we can rarely do anything about an error.

For example:

If a stack overflow occurs, an error will arise.

They are also ignored at the time of compilation.

Implementing Exceptions

- Surround an error prone code with ‘try’ block.
  - try{ some error prone code }

- Then specify the type of exception in the catch block, that you already know may occur, right after try block.
  - try{

```
        some error prone code
    }
    catch( FileNotFoundException ex){
        some more  code  to resolve or continue after error
    }
```

What if we have multiple known exceptions

Multiple Catch Blocks

- try{
  some error prone code
  }
  catch(Exception1 ex1){
  some more code to resolve or continue after error
  }
  catch(Exception2 ex2){
  some more code to resolve or continue after error
  }
  catch(Exception3 ex3){
  some more code to resolve or continue after error
  }

finally block

- The optional ‘finally’ block consists of the ‘finally’ keyword, followed by code enclosed in curly braces.
- If it’s present , it’s placed after the last catch block.
- If there are no catch blocks, the ‘finally’ block is required and immediately follows try block.

When the finally Block Execution

- The ‘finally’ block will execute whether or not an exception is thrown in the corresponding try block.

Practice Question

- Create a method name division() where you read two inputs from the user as ‘int’ type , then perform division of those numbers.
- Handle the exception where user could try to divide the number by zero (0)

(hint: ArithmeticException)

Question 2

- To the Employee class apply the exception to all the setter methods with their appropriate handling .

Learning

- Arrays
- Array List
- Enhanced for loop

Arrays

- Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same data type.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same data type.

Declaring Array Variables

- To use an array in a program, you must declare a variable to reference the array, and you must specify the data type of array the variable can reference.

Syntax:

- dataType[] arrayName; // preferred way. or

- dataType arrayName[]; // works but not commonly used.

Example

- double[] myArray; // preferred way. or
- double myArray[]; // works but not commonly used.

Creating Arrays

- You can create an array by using the new operator with the following syntax −

```
dataType[] arrayName = new dataType [arraySize];
```

For example:

```
        int[] myArray = new int[10];
// Create the “myArray”  Object.
```

Passing Arrays to Methods

To Pass an array arguments to a method, specify the name of the array without any bracket.

For Example,

if array hourlyTemperatures = new double[24];

Then the method call:

methodArray(hourlyTemperatures);

Passes the reference of array hourlyTemperatures to method methodArray

Contd.,

- For method to receive an array reference through method call, the method's parameter must specify an array parameter.

For Example:

```
void methodArray(double[]  b )
```

Demo code

```
 * @author Radhika
 */
public class PassArray {
    // Multiply each element of an array by 2
    public void doubleArrayElements(int[]array)
    {
        for (int counter = 0; counter < array.length; counter++){
            array[counter]*=2;
            }
    // multiply argument by 2
    public void modifyElement(int element){
        element *= 2;
   System.out.println("Value of element in modifyElement:\n"+element);
}
}
```

Main class

```
public class PassArrayTest {
// main class array and calls modifyArray and modifyElement
    public static void main(String[] args) {
        int[] array1 = {10,20,30,40,50};
        System.out.print("Effects of passing reference to entire array:"
        + "\n The values of the original array are:\n");
        // output original array elements
        for(int i=0;i<array1.length;i++){
            System.out.println(array1[i]);
        }// end of for loop
        ////////////////////////////////////////////////////////////
        PassArray psa = new PassArray();
        psa.doubleArrayElements(array1);// pass array refrence
        System.out.println("The values of the modified arrays are");
        //output modified array elements
        for(int j : array1){
            System.out.println(array1[j]);
        }    // end of for loop
        System.out.println("\n Effects of passing array element value:"
        +"\n array1[3] before modifyElement:"+ array1[3]);

        psa.modifyElement(array1[3]);// attempt to modify array[3]
        System.out.println("\n array[3] modified after modifyElement:"
                                    +array1[3]);}}
```

ArrayList

- Generally we all know that, Java arrays are of a fixed length. **After arrays are created, they cannot grow or shrink**, which means that you must know in advance how many elements an array will hold.
- Array lists are created with an initial size. **When this size is exceeded, the collection is automatically enlarged**. When objects are removed, the array may be shrunk.

Why ArrayList is better than Array?

- The limitation with array is that it has a fixed length so if it is full you cannot add any more elements to it, likewise if there are number of elements gets removed from it the memory consumption would be the same as it doesn’t shrink.
- On the other ArrayList can dynamically grow and shrink after addition and removal of elements . Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy.

Syntax

- We can create an ArrayList by writing a simple statement like this:--

>> ArrayList<String> alist = new ArrayList<String>();

- This statement creates an ArrayList with the name alist with type “String”.
- The type determines which type of elements the list will have. Since this list is of “String” type, the elements that are going to be added to this list will be of type “String”.

Demo Code

```java
public class ArrayListDemo {
    public static void main(String[] args) {
        // TODO code application logic here
        //Arraylist declaration
        // ArrayList name " myList" of Integert type
        ArrayList<Integer> myList =  new ArrayList<Integer>(5);
        // You may  or may not define the size of an array list
        //Adding the elements to the array list myList
        myList.add(1);
        myList.add(15);
        myList.add(85);
        myList.add(45);
        // enhanced for statement/loop
        for(Integer x : myList){
            System.out.println("The elements of array list:" +x);}
        // to retereive the size of array list
        System.out.println("Size of array list:"+ myList.size());
            //To set the value of an arrayList at index 1
        System.out.println( myList.set(1,500));
        System.out.println("Replace the element of myList at index 1"+ myList.s
        for(Integer x : myList){
            System.out.println("The elements of array list:" +x);}
    }}
```

Enhanced ‘for’ statement

- The enhanced for statement iterates through the elements of an array without using a counter, thus avoiding the possibility of “stepping outside” the array.

- Counting details are hidden from you in the enhanced for statement:

Syntax

for(parameter : arrayName)

{

- Statement;

}

parameter has a type and an identifier (e.g., int number)

arrayName is the array through which to iterate.

Contd.,

```
for( int counter = 0; counter < arrayName.length; counter++)
{
total += array[ counter ]
}
```

The above for statement is hidden in the enhanced for statement.

Demo code, enhanced for statement

```
public class EnchanncedForLoopTest {
    public static void main(String[] args) {
        // TODO code application logic here
        int [] array = {87,68,94,100,83,78,85,91,76,87};
        int total =0;
           // add each element's value to total
        for(int number : array){
            total+=number;}
            System.out.println("Total of array elements :"+ total);
    }

}
```