Exception handling

- An exception is when a problem occurs
- Such a problem happens infrequently
- Allows the program to handle the problem instead of terminating the program

- Exception handling is a way of avoiding intermixing program logic and error-handling logic
- This improves
  - Code clarity
  - Modifiability

- Common errors handled are:
  - Out-of-range array subscripts
  - Arithmetic overflow (i.e. a value outside the representable range of values)
  - Division by zero
  - Invalid method parameters
  - Unsuccessful memory allocation

- Exceptions can be handled based on:
  - Exception types
  - Exception types that belong to a inheritance hierarchy

s21

- Avoid using exception handling to handle potential errors in the conventional flow of control
- Handling a large exception case can be
  - Cumbersome
  - Difficult to read
  - Difficult to maintain

s21

- Syntax

```
try {
   …
   throw exeptionType;
}

catch (exeptionType e){
   handling exception
}
```

s21

- The try block groups a number of sentences whereby an exception can be thrown
- An exception can also be thrown by a method inside the try block
- The catch block is in charge of handling an exception type

s21

- An exception can be re-thrown
  - In case an exception handler can only deal partially or cannot deal with the exception, it is passed to another handler

- When an exception is re-thrown
  - It is caught by the catch block at the next outer level
  - An exception can be re-thrown by using the statement *throw;*

Quiz: Test the following code

```
import java.util.Scanner;

class Math1a {
   public static void main(String[] param) {
      Scanner console = new Scanner(System.in);

      System.out.print("numerador: ");
      int n = console.nextInt();

      System.out.print("denominador: ");
      int d = console.nextInt();
   }
}
```

```
import java.util.Scanner;

class Math1b {
   public static void main(String[] param) {
      Scanner console = new Scanner(System.in);
      int n = 0;
      int d = 0;

      try{
         System.out.print("numerador: ");
         n = console.nextInt();

         System.out.print("denominador: ");
         d = console.nextInt();
      }
      catch(Exception e) {
         System.err.printf("\nExcepcion: %s\n", e);
         System.exit(-1);
      }
   }
}
```

- Quiz: make a program that reads two integers and divides them and prints out the result by using:
  - public static int divide(int n, int d);

- What happens when the denominator is zero?
- What happens if string is given as input instead of an integer?

- Quiz: include an exception handler to catch the exception Exception to handle the type mismatch of the input parameters

Handling input parameters

```
import java.util.InputMismatchException;
import java.util.Scanner;

class Math {
   public static void main(String[] param) {
      Scanner console = new Scanner(System.in);

      while(true) {
         try {
            //lee numerador y denominador
            //invoca divide()
            //imprime resultado
            break;
         } //try

         catch(Exception e) {
            System.err.printf("\nExcepcion: %s\n", e);
            console.nextLine();
            System.out.println("Debe introducir enteros. Intente de nuevo.");
         }//catch
      }//while
   }//main
}//class
```

- Quiz: include an exception handler to catch the exception InputMismatchException to handle the type mismatch of the input parameters

Handling input parameters

```
import java.util.InputMismatchException;
import java.util.Scanner;

class Math {
  public static void main(String[] param) {
    Scanner console = new Scanner(System.in);

    while(true) {
      try {
        //lee numerador y denominador
        //invoca divide()
        //imprime resultado
        break;
      } //try

      catch(InputMismatchException inputMismatchException) {
          System.err.printf("\nExcepcion: %s\n", inputMismatchException);
          console.nextLine();
          System.out.println("Debe introducir enteros. Intente de nuevo.");
      }//catch
    }//while
  }//main
}//class
```

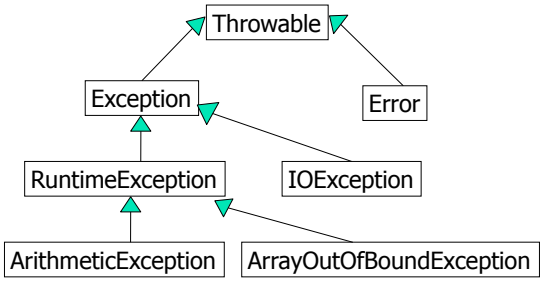
- Quiz: include an exception handler to catch the exception ArithmeticException to handle division by zero

Handling division by zero

```
catch(ArithmeticException arithmeticException) {
        //maneja error
      }//catch
```

```
public static int divide(int n, int d) throws ArithmeticException {
    return n/d;
  }
```

Part of the Throwable hierarchy



- The class Exception represents exceptional situations in a Java program
- The class Error represents exceptional situation in the JVM
  - Errors are not caught

- Verified Exceptions
  - The compiler requires exceptions are caught or declared (throws)
- Not verified Exceptions
  - The compiler does not require exceptions are caught or declared
  - All exceptions that inherit from RuntimeException are not verified
  - Examples: AritmeticException, ArrayIndexOutOfBoundsException

- If several catch blocks match an exception type, only the first one will be executed
  - Example: block 1 with Exception,block 2 with ArithmeticException: only block 1 is executed

Finally block

- Make sures that sentences within the block are executed indendently whether
  - an execption is caught or not
  - A break, return, or continue instruction is reached
- The finally block will not be executed if a System.exit call is reached

- It is placed after the last catch block
- In most cases used to free resources e.g. close files or close DB connections

```
try {
   ...
}
catch(...){
   ...
}

...
catch(...){
   ...
}
finaly{
   ...
}
```

```
public class UsoDeExcepciones {
  public static void main(String args[]) {
    try {
      lanzaExcepcion();
    }
    catch(Exception exception) {
      System.err.println("
          La excepcion se manejo en main");
    }

    noLanzaExcepcion();
  }
```

What is the output of this program?

```
public static void lanzaExcepcion() throws Exception {
    try{
      System.out.println("Metodo lanzaExcepcion()");
      throw new Exception(); //que pasa si coment esta linea?
    }
    catch(Exception exception) {
      System.err.println(
        "La excepcion se manejo en el metodo lanzaException()");
      throw exception; //que pasa si comentamos esta linea?
    }
    finally{
      System.out.println("Se ejecuto finally en lanzaExcepcion()");
    }
  }

  public static void noLanzaExcepcion(){

    try{
      System.out.println("Metodo noLanzaExcepcion()");
    }
    catch(Exception exception) { //no se ejecuta
      System.err.println(exception);
    }
    finally{
      System.out.println("Se ejecuto finally en noLanzaExcepcion()");
    }
    System.out.println("Fin del metodo noLanzaExcepcion()");
  }
}
```

```
public class UsoDeExcepciones2 {
  public static void main(String args[]) {
    try {
      lanzaExcepcion();
    }
    catch(Exception exception) {
      System.err.println("La excepcion se manejo en main");
    }
  }

  public static void lanzaExcepcion() throws Exception {
    try{
      System.out.println("Metodo lanzaExcepcion()");
      throw new Exception();
    }
    catch(RuntimeException runtimeException) {
      System.err.println("La excepcion se manejo en el metodo lanza Exception()");
    }
    finally{
      System.out.println("Se ejecuto finally en lanzaExcepcion()");
    }
  }
}
```

What is the output of this Program?

printStackTrace, getStackTrace, getMessage

```
public class UsoDeExcepciones3 {
  public static void main(String args[]) {
    try{
      metodo1();
    }
    catch(Exception exception) {
      System.err.printf("%s\n\n", exception.getMessage() );
      exception.printStackTrace();

      StackTraceElement[] elementosRastreo = exception.getStackTrace();

      System.out.println("\nRastreo de la pila de getStackTrace:" );
      System.out.println("Clase\t\t\tArchivo\t\t\tLinea\tMetodo");

      for( StackTraceElement elemento: elementosRastreo ) {
        System.out.printf("%s\t", elemento.getClassName());
        System.out.printf("%s\t", elemento.getFileName());
        System.out.printf("%s\t", elemento.getLineNumber());
        System.out.printf("%s\n", elemento.getMethodName());
      }
    }
  }
```

```
public static void metodo1() throws Exception {
    metodo2();
  }

  public static void metodo2() throws Exception {
    metodo3();
  }

  public static void metodo3() throws Exception {
    throw new Exception(
          "La excepcion se lanzo en metodo3");
  }
}
```

- Quiz: compile UsoDeExcepciones3.java and analyse the output of the program

Linked Exceptions

- Allows an exception object to keep complete track information
- The constructor of Exception uses two arguments
  - The seconde argument passes the current exception object

Linked Exceptions

```
public class UsoDeExcepciones4 {
  public static void main(String args[]) {
    try{
      metodo1();
    }
    catch(Exception exception) {
      exception.printStackTrace();
    }
  }

  public static void metodo1() throws Exception {
    try{
      metodo2();
    }
    catch(Exception exception) {
      throw new Exception(
        "La excepcion se lanzo en metodo1", exception);
    }
  }
```

```
public static void metodo2() throws Exception {
    try{
      metodo3();
    }
    catch(Exception exception) {
      throw new Exception(
            "La excepcion se lanzo en metodo2", exception);
    }
  }

  public static void metodo3() throws Exception {
    throw new Exception(
          "La excepcion se lanzo en metodo3");
  }
}
```

- Quiz: compile UsoDeExcepciones4.java and analyse the output of the program

Declaring new exception types

```
class ExcepcionListaVacia extends RuntimeException {
   public ExcepcionListaVacia() {
     super("Lista vacia");
   }
}

class Lista {
   Object Elementos[];
   int ultimo;

   public Object eliminarDelFinal() throws ExcepcionListaVacia {
     if( estaVacia() )
        throw new ExcepcionListaVacia();

     ultimo--;
     return Elementos[ultimo];
   }

   public boolean estaVacia() {
     return true;
   }
}
```

```
public class PruebaLista {
   public static void main(String args[]) {
     Lista lista = new Lista();

     try{
       lista.eliminarDelFinal();
     }
     catch(ExcepcionListaVacia excepcionListaVacia) {
       excepcionListaVacia.printStackTrace();
     }
   }
}
```

Assertions

- The instruction assert
  - Evaluates a boolean expresion
- Two forms of the assert expression:
  - assert expresion;
  - Evaluates expresion and throws an AssertionError exception if it evalutes to false

- assert expresion1 : expresion2;
- Evaluates expresion1 and throws an AssertionError exception if it evalutes to false
- expresion2 is used as a message error
- Example:
  - assert(n>=0 && n<=10) : "invalid number: " +n

- Compiling:
  - javac **-source 1.4** MyClass.java
- Enabling when running
  - java –ea MyClass
- Disabling when running
  - java MyClass

- Can be useful to define
  - Preconditions
  - Postconditions
  - Invariants
- Assertions reduce performance
  - Must not be used to validate runtime errors
  - Runtime errors must be validated with exceptions
  - Assertions are used for debugging and testing

- Preconditions
  - Check valid values for input parameters of a method
  - Do not use assertions to verify preconditions in public methods use exceptions instead
  - If assertions disabled, preconditions would not be validated
  - Assertions can be used in nonpublic methods to detect library errors

- Postconditions
  - Check valid values for a return value or an output parameter of a method
  - Can be used in public and nonpublic methods
  - Can be used in public methods since a postcondition validates the library behaviour instead of runtime input parameters

- Invariants
  - Validate invariants in *if* and *switch* statements

```
if( x== 4)
   ...
else {
   assert (x==2)  : x; //we know x must be 2
   ...
}
```

```
switch(y) {
   case 1: ...
   case 2: ...
   default:
      throw new AssertionError(y);
      //assert(false):y;
}
```

- Invariants
  - Validate control-flow: execution should never reach this point

```
void foo() {
  for (...) {
     if (...)
        return;
  }
  assert(false); // Execution should never reach this point!
}
```

Precondition

```
import java.util.Scanner;

public class Rendering {
   public static final double MAX_REFRESH_RATE = 100000.0;
   public static double interval;

   public static void setRefreshRate(double rate) {
      // Enforce specified precondition in public method
      if (rate <= 0 || rate > MAX_REFRESH_RATE)
         throw new IllegalArgumentException("Illegal rate: " + rate);

      setRefreshInterval(1000/rate);
      System.out.println("Refresh interval: " + 1000/rate);
   }

   private static void setRefreshInterval(double x) {
      // Confirm adherence to precondition in nonpublic method
      assert !(x <= 0 || x*1000 > MAX_REFRESH_RATE) : interval;

      interval = x;
   }
```

```
public static void main(String args[]) {
      Scanner console = new Scanner(System.in);
      System.out.print("rate: ");
      int n = console.nextInt();
      setRefreshRate(n);
   }
}
```

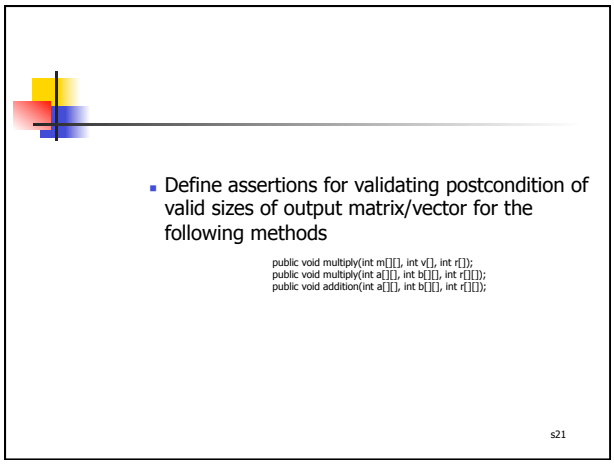
- Activity:
  - Define an exception handler for the exception ArrayIndexOutOfBoundException for the following methods

```
public void print(int v[]);
public void print(int m[][]);
public void multiply(int m[][], int v[], int r[]);
public void multiply(int a[][], int b[][], int r[][]);
public void addition(int a[][], int b[][], int r[][]);
public int greater(int m[][]);
```

  - Define an exception handler for validating precondition of valid sizes of input matrix/ vector for the following methods

```
public void multiply(int m[][], int v[], int r[]);
public void multiply(int a[][], int b[][], int r[][]);
public void addition(int a[][], int b[][], int r[][]);
```

s21



- Define assertions for validating postcondition of valid sizes of output matrix/vector for the following methods

```
public void multiply(int m[][], int v[], int r[]);
public void multiply(int a[][], int b[][], int r[][]);
public void addition(int a[][], int b[][], int r[][]);
```

s21