**UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA**
**FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)**

# Formato para prácticas de laboratorio

| CARRERA | PLAN DE ESTUDIO | CLAVE DE UNIDAD DE APRENDIZAJE | NOMBRE DE LA UNIDAD DE APRENDIZAJE |
|---|---|---|---|
| Ingeniero en Computación | 2009-1 | 12099 | Programación Orientada a Objetos |

| PRÁCTICA No. | LABORATORIO DE | Programación Orientada a Objetos | DURACIÓN (HORAS) |
|---|---|---|---|
| 10 | NOMBRE DE LA PRÁCTICA | Manejo de Excepciones | 2 |

## 1. INTRODUCCIÓN

Java provee un mecanismo para identificar situaciones anormales, como errores, que de no atenderse pueden provocar que un programa funcione inesperadamente. La importancia de atender o manejar excepciones se puede ilustrar con el siguiente ejemplo.

Algunos sistemas críticos como los de los bancos deben ser muy seguros debido a que su mal funcionamiento puede ocasionar perdidas tanto para los bancos como para los cuentahabientes. Por lo tanto si el software de un banco se encuentra un error, no es apropiado que unicamente muestre un error en pantalla y termine. Primeramente, se debe de intentar evitar el error, pero si el error es inevitable, el software debe proveer un mecanismo para almacenar la información que se estaba manejando al momento del error para así poder recuperarla.

Las excepciones son un mecanismo que permite atender situaciones de error y tomar medidas apropiadas para que los programas puedan recuperarse de los errores o en caso extremo, salvaguardar la información del estado del programa.

| Formuló | Revisó | Aprobó | Autorizó |
|---|---|---|---|
| Cecilia Curlango Rosas<br>Maria Luisa Gonzalez Ramirez | Gloria E. Chavez Valenzuela | | M.C. Maximiliano de las Fuentes Lara |
| Nombre y Firma del Maestro | Nombre y Firma del Responsable de Programa Educativo | Nombre y Firma del Responsable de Gestión de Calidad | Nombre y Firma del Director de la Facultad |

## 2. OBJETIVO (COMPETENCIA)

Desarrollar programas que atrapen y manejen excepciones para atender situaciones de error dentro de los programas demostrando una actitud creativa y responsable.

## 3. FUNDAMENTO

### Exceptions.

Una excepción en Java es un evento que ocurre durante la ejecución de un programa y que interrumpe el flujo normal del programa. Cuando ocurre un error dentro de un método este crea un objeto y se lo envía al sistema de ejecución. El objeto es llamado un objeto *Exception* y contiene información sobre el error. Lanzar una excepción se le llama al hecho de crear una excepción y lanzarla al sistema de ejecución.
Después de que el método lanza la excepción el sistema trata de encontrar algo para manejarla.
Algunas veces no queremos hacernos cargo de la excepción en el método que ocurre. Si ese es el caso podemos lanzarla fuera del método para que otro método se haga cargo de ella. El sistema de ejecución busca el método que contiene el código para manejarla.

Usar excepciones para manejar los errores tiene ventajas sobre la forma tradicional del manejo de errores como por ejemplo:

a) Separa el código para manejar el error del código normal del programa
b) Se puede propagar el error a otro método para que se haga cargo de él.
c) Se agrupa y diferencia los tipos de errores

**Tipos de Exceptions.**

`RuntimeException`. Son errores en tiempo de ejecución como una división entre cero o el acceso a el indice mas allá de las dimensiones de un arreglo.
`IOException`. Errores de entrada/salida de datos.

Pero también nosotros podemos crear y lanzar nuestras propias excepciones.

Existen 3 bloques principales para manejar las excepciones en java.

**Bloque *try*.**

En el bloque `try` se pone todo el código que pueda generar una excepción. La sintaxis de este bloque es:

```
try{
//codigo que genera excepciones
}
```

El bloque `try` debe ir seguido, al menos, por una cláusula `catch` o una cláusula `finally.`

**Bloque *catch***

El bloque `catch`, es el código que se ejecuta cuando se produce la excepción. En este bloque tendremos que asegurarnos de colocar código que no genere excepciones. Se pueden colocar sentencias `catch` sucesivas, cada una controlando una excepción diferente. No debe intentarse capturar todas las excepciones con una sola cláusula. La sintaxis de este bloque es:

```
catch( tipoExcepcion e ) {
...
}
```

**Bloque *finally***

El bloque `finally` es el bloque de código que se ejecuta siempre, haya o no excepción. Este bloque `finally` puede ser útil cuando no hay ninguna excepción. Es código que se ejecuta independientemente de lo que se haga en el bloque `try`. La sintaxis de este bloque es:

```
finally{
….
}
```

Cuando manejamos una excepción, debemos decidir qué acciones vamos a tomar. En la mayoría de los casos, bastará con presentar una indicación de error al usuario y un mensaje avisándole que se ha producido un error y que decida si quiere o no continuar con la ejecución del programa.

### La sentencia *throw*.

La sentencia `throw` se utiliza para lanzar objetos tipo `Throwable`. Por ejemplo:

```
throw new Exception(“Mensaje de error”);
```

Al lanzar una excepción ocurren diferentes cosas:

- Se sale inmediatamente del código actual.
- Si el bloque tiene asociado una clausula `catch` adecuada para la excepción generada se ejecuta el bloque `catch`.
- Si no, se sale inmediatamente del bloque dentro del cual esta el código que produjo la excepción y se busca el `catch` adecuado.
- El proceso continua hasta llegar al método `main` de la aplicación.

### Propagación de excepciones (*throws*)

Si un método lanza una excepción debemos agregar a la firma del método una clausula `throws` que incluya la lista de los tipos de excepciones que se pueden producir al invocar el método.

Por ejemplo:

```
public tipoRetorno nombreMetodo() throws tipoException{
…
}
```

Un ejemplo del manejo de excepciones se muestra a continuación.
En el siguiente código se muestra una clase que contiene un método estático que realiza una división.

**LISTADO 1**

```
class Operaciones {
```

```
    public static double division_real (double dividendo, double divisor) {
              return (dividendo / divisor);
    }
}
```

Para comprobar el funcionamiento del código del Listado 1 ejecutamos el código del Listado 2

**LISTADO 2**

```
public static void main (String [] args) {
  double x = 15.0;
  double y = 3.0;
  System.out.println ("El resultado de la división real de " + x + " entre " + y +
" es " + division_real (x, y));
 }
```

El resultado de compilar y ejecutar el anterior programa sería:

**El resultado de la división real de 15.0 entre 3.0 es 5.0**

El código del Listado 1 y 2 funciona correctamente. Pero hay una situación que no hemos considerado. ¿Qué pasaría si el divisor fuera 0?

**LISTADO 3**

```
public static void main (String [] args){
  double x = 15.0;
  double y = 0.0;
  System.out.println ("El resultado de la division real de "+ x + " entre " + y +
" es " + division_real (x, y));
 }
```

Al ejecutar la anterior operación se obtiene como resultado:

**El resultado de la division real de 15.0 entre 0.0 es Infinity**

Este comportamiento no es el deseado por nosotros, el valor "`Infinity`" podría provocar respuestas inesperadas, así que puede que un usuario quiera tratar dicho caso "excepcional" de una forma diferente. Para esto debemos definir una estructura condicional (un bloque "`if ... else ...`") y utilizar la sentencia `throws` para lanzar una excepción .
Modificamos el listado 1 para que lance una excepción si el divisor es igual a 0.

**LISTADO 4**

```
class Operaciones {
  public static double division_real (double dividendo, double divisor) throws
ArithmeticException {
      if(divisor==0.0) {
          throw new ArithmeticException("El divisor no puede ser cero");
      }
```

```
      return (dividendo / divisor);
    }
}
```

Y también debemos modificar el Listado 3 para que atrape la excepción que se puede generar en el método *division_real()*.

**LISTADO 5**

```
public static void main (String [] args){
  double x = 15.0;
  double y = 0.0;
try{
  System.out.println ("El resultado de la division real de "+ x + " entre " + y +
" es " + division_real (x, y));
}catch(ArithmeticException e)
{System.err.println("Error en la division"+e.getMessage());
}

}
```

Todo sobre excepciones lo puedes encontrar en:

http://download.oracle.com/javase/tutorial/essential/exceptions/definition.html

## 4. PROCEDIMIENTO (DESCRIPCIÓN)

1. El método `getPixelMatix()` que se agregó a la clase `Picture` regresa una imágen representada en una matriz de pixeles. Si se intenta acceder a una coordenada de esta matriz que está fuera de límite, Java genera una excepción tipo `ArrayIndexOutOfBounds`. Modifique el código del método `getPixelMatrix()` para que si tratan de acceder a una coordenada que no exista lance una excepción de su propia creación. El nombre de la excepción será `CoordenadaDeImagenFueraDeRangoException`.

2. Escriba un programa sencillo en el que demuestre el funcionamiento de la `excepción CoordenadaDeImagenFueraDeRangoException` que agregó. Demuestre en el código cómo generó, atrapó y atendió la excepción.

3. Modificar la clase `Turtle` para que lance una excepción `PosicionFueraDeRangoException` cuando se intente mover a la tortuga a una posición que quede fuera del mundo.

4. Escriba un programa sencillo en el que demuestre el funcionamiento de la excepción `PosicionFueraDeRangoException` que agregó. Demuestre en el código cómo generó, atrapó y atendió la excepción.

| A) EQUIPO NECESARIO | MATERIAL |
| --- | --- |

Computadoras con capacidad para ejecutar el entorno de desarrollo Netbeans. Paquete `misClases`. Al menos una imagen para manipularla con los métodos presentados y desarrollados en la practica.

## 7. REFERENCIAS


Netbeans
http://netbeans.org/downloads/
Java 6
http://www.oracle.com/technetwork/java/javase/downloads/index.html