Objetivos:

- Tipos Genéricos
- Excepciones, Declaración y propagación (cláusula *throws*)
- Captura, Manejador de excepciones.
- Incorporar información a excepciones.
- Introducción a las IU. División en capas.

En la práctica usaremos algunas clases definidas en las prácticas anteriores. Es por ello que se recomienda finalizar dichas prácticas antes de comenzar con la presente.

Ejercicio 1.
Una fábrica de juegos de PC desea modelar un sistema de selección aleatoria de Jugadores (**RandomSelector**). Dado que la selección aleatoria se usa en muchos y variados juegos (p.e: selección aleatoria de números, de cartas, de colores, de fichas, etc.) se detalló que el RandomSelector debería ser diseñado de tal forma que se pueda usar con cualquier tipo de objetos.

Nota: *El funcionamiento debería ser el siguiente: Una vez creado el **RandomSelector** con el tipo de Objeto que va a seleccionar este permanece así. Al RandomSelector se le pueden agregar **N** objetos a seleccionar. Cuando se le dice selectNext() retorna un objeto seleccionado de manera aleatoria.*

Ejercicio 2.
Dado el siguiente método:

```
public int dividir (int a, int b){
        return a/b;
  }
```

a) Realice pruebas con diferentes valores de a y de b.
b) ¿Qué pasa cuando b es 0?
c) Modifique el método para que ahora cuando b sea 0, el valor de retorno sea 0.
d) ¿Cuál sería la excepción más adecuada para utilizar en este caso?

Nota: en el ejercicio d) debe capturar la excepción con un **try - catch**

Ejercicio 3.
Cree la clase ***ColaDeTrabajo*** que permita encolar diversos trabajos. O sea, los trabajos a encolar deben implementar la interfaz ***Trabajo***. Defina en la clase ***ColaDeTrabajo*** un método *sacar()* que retorna el próximo trabajo a procesar. Además, agregue en dicha clase los atributos *nombre* y *lista* que representan el nombre de la cola y si esta lista o

no para retornar trabajos. Tenga presente, que cuando no existan trabajos en la cola o cuando la misma no esté lista se debe lanzar las siguientes excepciones: ***NoListaException*** y ***SinTrabajoEnColaException***

A continuación, se detalla cada una de las excepciones lanzadas por el método *sacar*() de ***ColaDeTrabajo***

```
public class NoListaException extends Exception {

	private String nombre;
	private long cantidadTrabajos;

	public NoListaException (String nom, long s) {
		nombre = nom;
		cantidadTrabajos = s;
	}

	@Override
	public String getMessage() {
		return "La Cola de Trabajo: " + nombre + " no está
		disponible. Cantidad de trabajos a procesar : " +
		cantidadTrabajos;
	}
}


public class SinTrabajoEnColaException extends Exception {

	private String nombre;

	 public SinTrabajoEnColaException (String nom) {
		nombre = nom;
	 }

	 @Override
	 public String getMessage() {
		return "La cola " + nombre + " no tiene trabajos para
		procesar. ";
	 }
}
```

Se pide:

a) Implementar la clase ***ColaDeTrabajo*** y definir el método `sacar()` en dicha clase.
b) ¿Cómo se lanzan las excepciones anteriores dentro del método?
c) ¿Cómo se captura las excepciones al llamar al método `sacar()`?

Ejercicio 4.

Modifique la clase ***RandomSelector*** del Ejercicio 1 para lanzar la excepción ***IndexOutOfBoundsException*** al momento de solicitar un elemento del selector y no disponer de ninguno en su colección para escoger.

Ejercicio 5.

Extienda el Ejercicio 2 de la Práctica 2 para que ahora, cuando se supere la cantidad de pasajes vendidos para ese vuelo, se genere la excepción ***SuperaPasajesVendidosException***. También, en caso de que un pasajero quiera comprar más de un pasaje por Vuelo se deberá lanzar la excepción ***PasajeVendidoAPasajeroException***. Genere el código necesario para capturar dichas excepciones.

Ejercicio 6.

Defina en Java la clase ***DataBag***, la cual tiene un número máximo de elementos y permite almacenar **cualquier tipo** de objetos.

a) Implemente en JAVA la clase *DataBag*.
b) Defina el método *add*() para permitir agregar elementos a la bolsa y en el caso de que la misma este llena dispare la excepción ***FullDataBagException***.
c) Defina el método *remove*() para remover elementos de la bolsa y en el caso de que la misma este vacía se dispare la excepción ***EmptyDataBagException***

Nota: Tenga en cuenta que las dos excepciones ***FullDataBagException*** y ***EmptyDataBagException*** son excepciones chequeadas que deben ser creadas por usted como subclase de ***Exception***.

Ejercicio 7.

Dada la clase **PruebaExcepcion:**

```
public class PruebaExcepcion {

	public static void main(String st[]){
		PruebaExcepcion t1 = new PruebaExcepcion();
		t1.metodo(5,0);
	}
	public void metodo(int a, int b){
		try {
			int c = a/b;
```

```
                System.out.println("Después de la división");
            } catch (ArithmeticException ae) {
                System.out.println("Excepción Aritmética");
            }
            catch (Exception e) {
                System.out.println("Otra Excepción");
            }
            finally {
                System.out.println("Bloque Finally");
            }

            System.out.println("Después del bloque finally");
        }
}
```

a) ¿Qué retorna su ejecución?
b) ¿Qué hace la cláusula ***finally***?
c) Proponga ejemplo de casos en donde se podría llegar a usar la cláusula ***finally***.

Ejercicio 8.

Use el ejercicio 3 de la Práctica 1, modifique las clases que considere para que lancen las excepciones:

- ***SuperaLimiteMinimoException***
- ***SuperaCantidadExtraccionesException***

Interfaces de Usuario

*La **interfaz gráfica de usuario**, conocida también como GUI (del inglés graphical user interface) es un programa informático que actúa de **interfaz de usuario**, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.*

Ejercicio 9. – *Nuestro primer programa Swing* –

En este ejercicio se pide crear nuestra primera interfaz gráfica en Swing, haciendo el habitual *Hola, Mundo*. El resultado de ejecutar esta aplicación es la siguiente ventana:

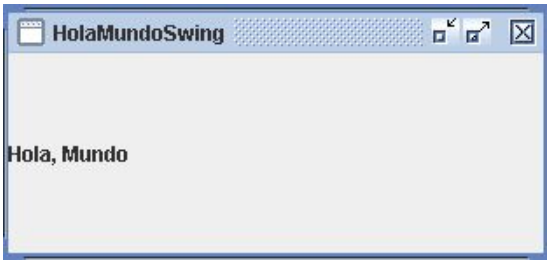

Figura 1

Ejercicio 10 – *Cronometro*

Se desea modelar un cronómetro como el de la figura 2. Este cronómetro es especial y solo mide intervalos de tiempo. Es decir, el tiempo transcurrido desde que se pulsó el botón Empezar hasta que se pulsa Detener.

Cuando se pulsa el botón Empezar se comienza a contar el tiempo (el mismo no se muestra). Cuando se pulsa el botón detener, el cronómetro deja de contar el tiempo y muestra el mismo en pantalla. El botón cambia su *label* acorde a su estado. El mismo tiene dos estados: Iniciado y Detenido.

Figura 2

Cuando el cronómetro está detenido y se pulsa Empezar, el valor anterior vuelve a 0.

Nota: *en java hay un método System.currentTimeMillis() que retorna el tiempo actual en milisegundos.*

Nota 2*: En este ejercicio se debe utilizar un modelo de dos capas. Por un lado, la GUI (y sus controllers) y por otro lado el Modelo (o Model). El modelo debe ser el objeto Tiempo que debe ser implementado con la lógica de la aplicación.*

Ambas capas deben relacionarse mediante algún patrón de diseño. Discuta con el ayudante el patrón a utilizar.