


Lenguaje de programación Java SE 8



Módulo 2 - Clase #02
Carrera Java Programmer SE 8



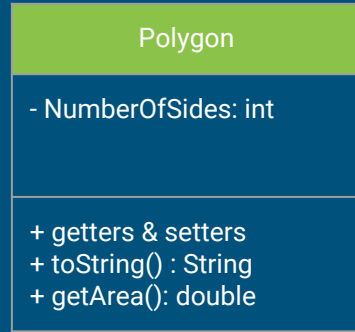
Miembros estáticos de una clase

- También se les conoce como miembros de clase. Son elementos que pertenecen a la clase como tal.
- Al momento de instanciar una clase, se monta en la memoria de trabajo una referencia, cuyo nombre almacena un objeto del tipo creado. Este contiene una copia de la estructura de la clase, incluyendo sus atributos y métodos, excluyendo los elementos estáticos.
- En resumen, los elementos estáticos pertenecen a la clase, no a los objetos.

Práctica

Súper Clase

abstract

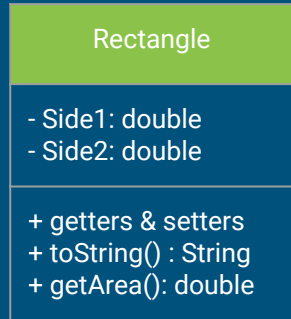


abstract

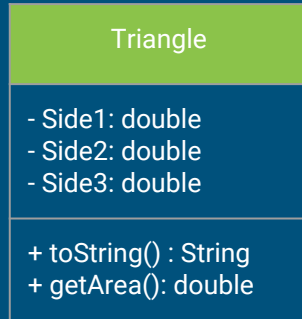
Indicaciones

Hacer un programa para calcular el área de polígonos (Triángulos y Rectángulos). El programa debe permitir recibir los datos de triángulos y rectángulos para luego, mostrar el área calculada. Para realizar este ejercicios se deben seguir las siguientes pautas:

1. Crear los siguientes paquetes:
 - a. com.main
 - b. com.polygon
 - c. com.triangles
 - d. com.rectangles
2. Crear una clase ejecutable dentro de com.main llamada Polymorphism2.
3. Definir una super clase abstracta llamada **Polygon** dentro de com.polygon.
4. Definir una sub clase llamada **Rectangle** dentro de com.rectangle.
5. Definir una sub clase llamada Triangle dentro de com.triangle.



Sub Clase



Sub Clase

Rectángulo: $A = a \times b$

Triángulo: $A = \frac{1}{2} (b.h)$

Clases Anidadas

Tal y como lo indica su nombre, es una clase definida dentro de otra. También se les conoce como clases internas.

Se usan para:

- Acceder a campos privados desde otra clase.
- Ocultar una clase de otras dentro de un mismo paquete.
- Crear clases internas anónimas.
- Gestionar eventos y retrollamadas.
- Acceder a los atributos ejemplares de otra clase.

```
package com.main;

public class ClaseExterna {

    class ClaseInterna {
        // Código de la clase interna
    }

    // Código de la clase externa
}
```

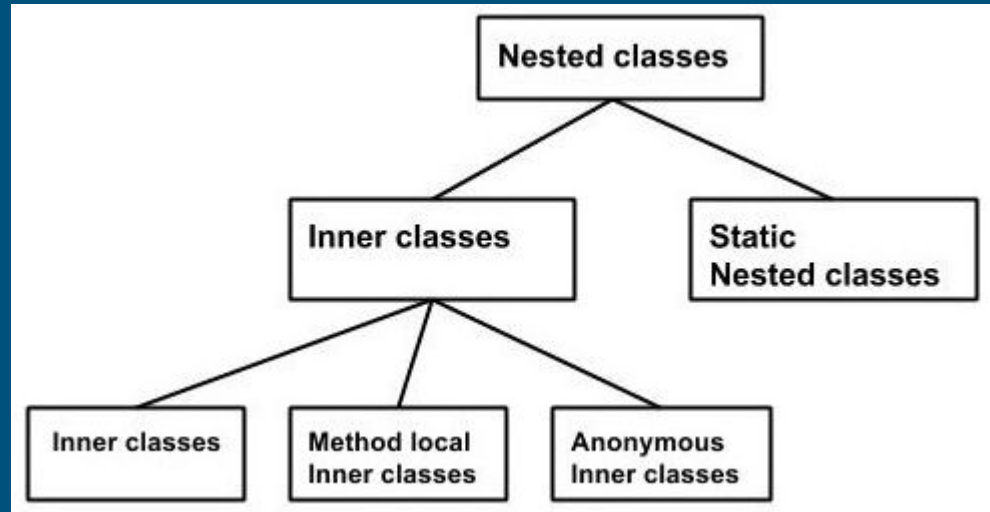
Clases Anidadas

Existen dos tipos de clases anidadas:

- Estáticas internas: se refiere a los miembros estáticos de una clase.
- No-Estáticas internas: se refiere a los miembros no estáticos de una clase.

A su vez, se dividen en:

- Internas de miembro.
- Internas de método local.
- Internas anónimas.



Clases Anidadas

Estáticas internas:

- Pueden acceder a métodos y atributos de la clase "externa", siempre que estos sean estáticos (incluidos privados).
- Pueden acceder a métodos y atributos de la clase

"externa", siempre que estos sean estáticos (incluidos privados).

- En caso de que los atributos o métodos no sean estáticos, se debe instanciar la clase externa.
- Para crear una instancia de la clase estática interna:
 - ClaseExterna.ClaseEstaticaInterna nombre = new ClaseExterna.ClaseEstaticaInterna();

```
package com.classes;

public class External { // Clase externa

    public static class Internal {

        public void method() { // Clase interna
            System.out.println("Hola desde la clase interna.");
        }

    }

}
```

Clases Anidadas

Internas de miembro:

- Es un elemento más de la clase externa.
- Puede acceder a elementos públicos y privados de la clase externa.
- Accede a elementos de la clase externa luego de instanciarla.
- No podemos definir variables ni métodos estáticos (salvo constantes).
- Para instanciar la clase Interna:
 - ClaseExterna ce = new ClaseExterna();
 - ClaseExterna.ClaseInterna ci = ce.ClaseInterna();

```
package com.classes;

public class External {// Clase externa
    public int attr1;

    private class Internal1 {
        public Internal1() {
        }
    }

    public class Internal2 {
        public Internal2() {
        }
    }
}
```


Clases Anidadas

Internas de método:

- La clase solo podrá ser instanciada dentro del mismo método.
- Una clase interna de método puede acceder a las variables locales del método.
- Los modificadores permitidos por este tipo de clase son "abstract" y "final".

```
package com.classes;

public class External {// Clase externa

    public void myMethod() {
        int number = 10;

        class MethodInnerClass {
            public void showNumber() {
                System.out.println(number);
            }
        }

        // Instanciamos la clase interna
        MethodInnerClass mic = new MethodInnerClass();
        mic.showNumber();
    }
}
```

Clases Anidadas

Internas anónimas:

- Siempre debe ser una subclase de otra clase ya existente o bien, implementar alguna interfaz.
- La definición de la clase anónima se lleva en una línea de código, por lo tanto se debe añadir punto y coma ";" al final de la declaración.
- Solamente podrá acceder a los métodos de la clase que se hayan heredado, sobrescrito o implementado.
- Es posible encontrarse una clase anónima como argumento de un método.

```
// Clase anónima
private interface AnonInnerClass {
    void anonMethodClass();
}

// Método para mostrar método de clase anónima
public void innerAnonClassMethod() {
    AnonInnerClass aic = new AnonInnerClass() { // Clase anónima

        @Override
        public void anonMethodClass() {
            System.out.println("Esto es el texto de un método de una clase anónima.");
        }

    }; // Fin clase anónima

    aic.anonMethodClass();
}
```

Ejemplo



Date / Time API

Información:

- Es inmutable y no tiene métodos setter.
- Mejor distribución de los métodos para operaciones con fechas, evitando la uniformidad aburrida.
- Simplificación de manejo de la zona horaria.

Clases:

- `java.time.LocalDate`
- `java.time.LocalDateTime`
- `java.time.LocalTime`
- `java.time.Period`

Métodos:

- `now`
- `isBefore`
- `isAfter`
- `parse`

Ejemplo



Interfaces

- Declaración formal de un contrato.
- Son abstractos, es decir, no poseen implementación.
- Uso de la palabra reservada “implements”.
- Podemos implementar múltiples interfaces a una clase.

Estructura de una Interface

Definición de una interface en Java:

```
<modificadores> interface <nombre_interface> [extends <interface padre>]
{
    <atributos>
    <métodos>
}
```

Uso de una interface en Java:

```
<modificadores> class <nombre_clase> [extends <superclase>] [implements
<interfacel,interface2,etc>]
{
    <implementar_métodos_interface>
}
```

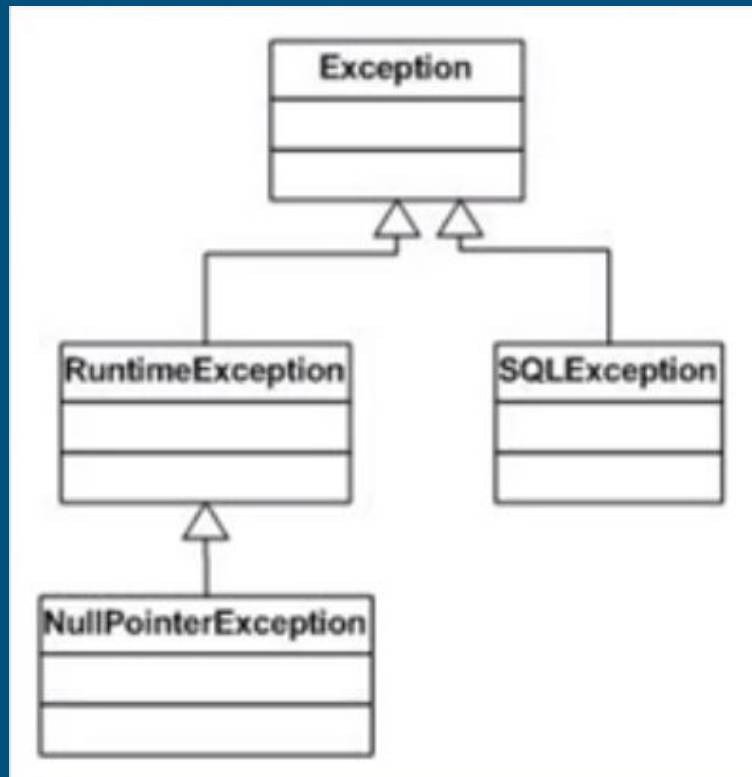
Excepciones

Check Exception

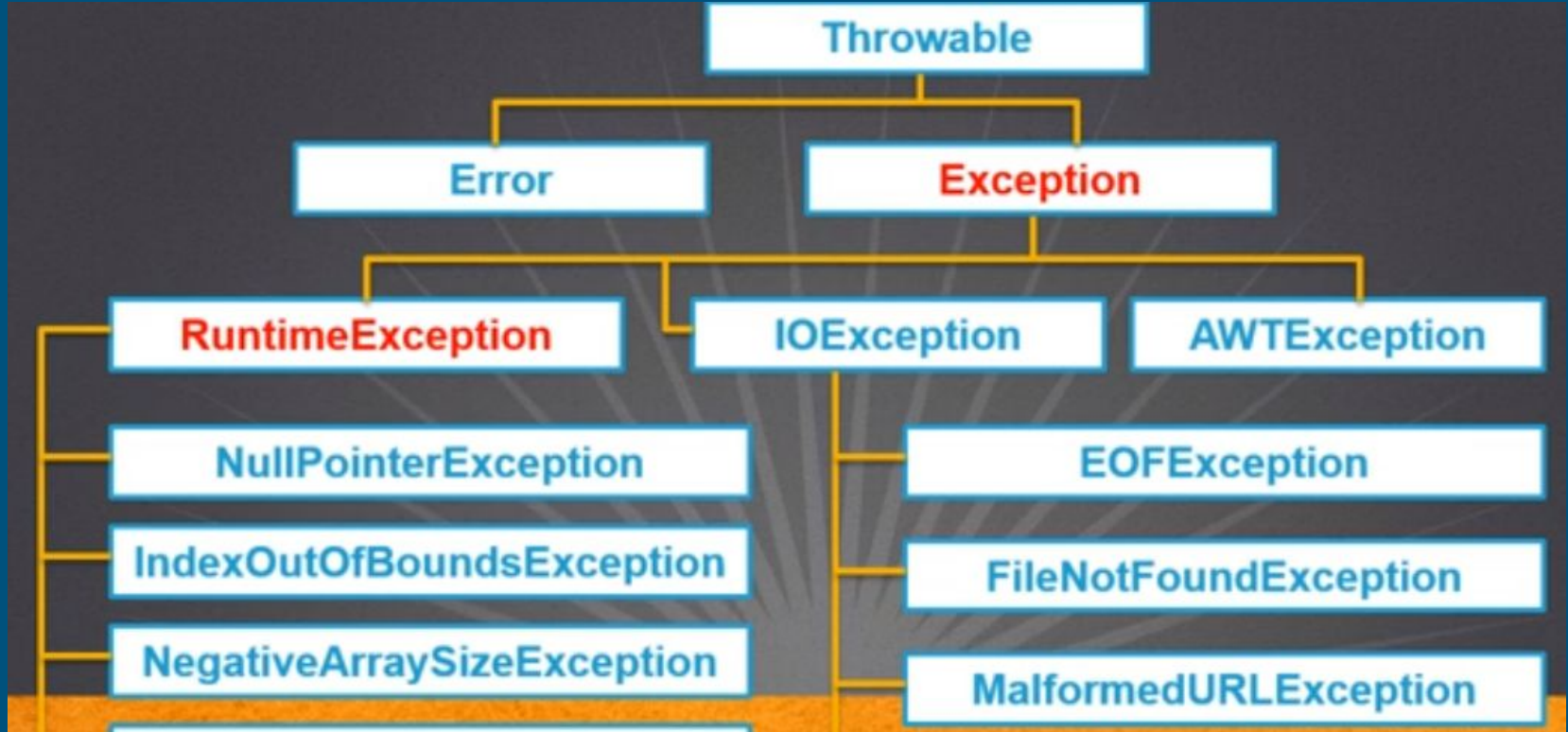
- Hereden de la clase Exception.
- Es obligatorio tratarlas.
- SQLException.

Unchecked Exception

- Hereden de la clase Exception.
- Es opcional tratarla.
- RuntimeException.
 - NullPointerException.

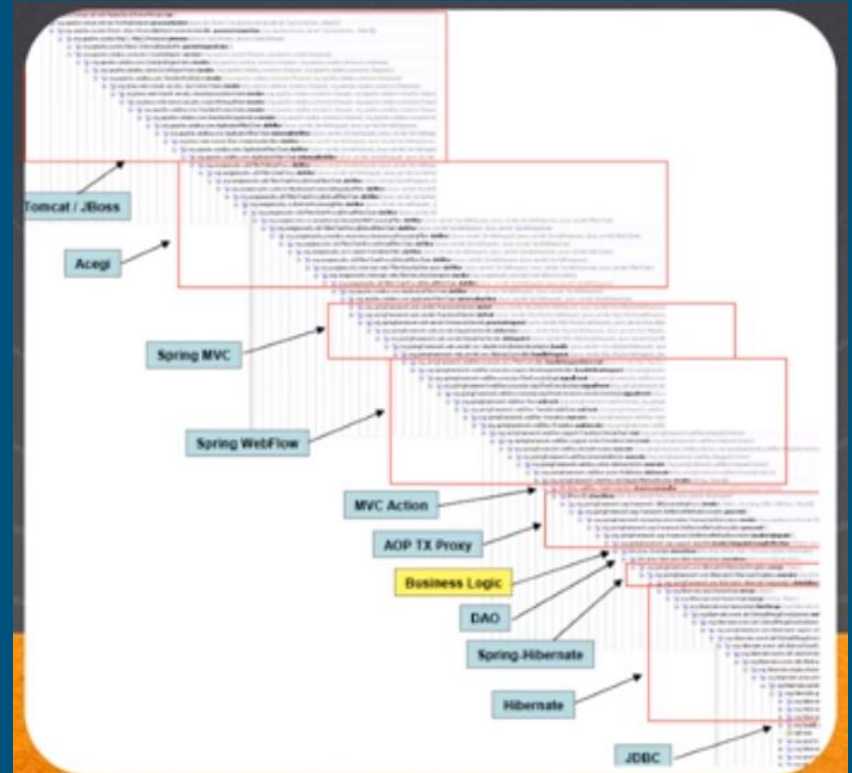


Excepciones más comunes



Stacktrace en Java

- Pilas de errores.
- Traza del error del inicio al fin.
- Flujo del error:
 - Método arroja un error.
 - Si no atrapa la excepción, la propaga hasta que alguna clase la maneja.
 - En caso de no manejarla el método main se satura
 - El programa se finaliza de manera anormal.



Manejo de Excepciones

```
public void verificaExcepciones() {  
    try {  
        // código que lanza excepciones  
    } catch (Exception ex) {  
        //Bloque de código que maneja la excepción  
        ex.printStackTrace();  
    }  
    finally{  
        //Bloque de código opcional, pero  
        //que se ejecuta siempre  
    }  
}
```

Cláusula Throws

```
public class ArrojarExcepcion {  
  
    public void metodoX() throws Exception {  
        throw new Exception("Mensaje de error");  
    }  
}
```

```
public class TestArrojarExcepcion {  
  
    public static void main(String args[]) throws Exception {  
        ArrojarExcepcion ae = new ArrojarExcepcion();  
        ae.metodoX();  
    }  
}
```

Creación de Nuestras Excepciones

```
public class MiExcepcion extends Exception{  
  
    public MiExcepcion(String mensaje){  
        super(mensaje);  
    }  
}
```

```
public class ArrojarExcepcion2 {  
  
    public void metodoX() throws MiExcepcion {  
        throw new MiExcepcion("Mi mensaje de error");  
    }  
}
```

Ejemplo

