Introducción a Java

//Parte 2

IT BOARDING

BOOTCAMP

Índice

01 Herencia

02 Clase abstracta

03 Polimorfismo

IT BOARDING
BOOTCAMP

TEMA

//

Programación orientada a objetos, ahora si

IT BOARDING

BOOTCAMP

Todos hemos estudiado que las clases heredan de otras clases. Pero, realmente ¿para qué?

IT BOARDING

BOOTCAMP

Herencia

La relación de herencia se identifica como **“es una especie de”**

Teléfono Celular es una especie de **Teléfono, Paloma** es una especie de **Ave**

En Java la herencia se indica con la palabra **extends**

Clases object

En Java, todas las clases (tarde o temprano) heredan de **Object.** Entre otros, Object declara los métodos **toString y equals**

```
Numero n = new Numero(5);
System.out.println(n); // que muestra?

Numero m = new Numero(6);
System.out.println(m); // que muestra?
```

¿Cuál será la salida de este programa?

Sobreescritura 1

Sobreescribir es **reescribir** un método que estamos heredando.

```
public class Numero
{
   private int valor;

   @Override
   public String toString()
   {
      return Integer.toString(valor);
   }

   // ...
}
```

Sobreescritura 2

¿Qué sucede al comparar las cadenas s1 y s2? ¿Y los números n y m?

```
String s1 = “Hola”;
String s2 = “Hola”;

if( s1 == s2 )
{
   // ...
}

if( s1.equals(s2) )
{
   // ...
}
```

```
Numero n = new Numero(5);
Numero m = new Numero(5);

if( n == m )
{
   // ...
}

if( n.equals(m) )
{
   // ...
}
```

Sobreescritura 3

En este caso, sobreescribiremos el método **equals**

```
public class Numero
{
   private int valor;

   @Override
   public boolean equals(Object o)
   {
      return this.valor==((Numero)o).valor;
   }

   // ...
}
```

Poliformismo 1

Los objetos nunca olvidan a qué clase pertenecen.

```
Object o1 = new Numero(5);
Object o2 = “Hola”;
Object o3 = new Date();

System.out.println(o1); // cuál será la salida?
System.out.println(o2); // cuál será la salida?
System.out.println(o3); // cuál será la salida?
```

Polimorfismo 2

¿Qué hace la función mostrarArray? ¿Funciona correctamente?

```
Object[] arr = new Object[3];
arr[0] = new Numero(5);
arr[1] = “Hola”;
arr[2] = new Date();

mostrarArray(arr);
```

```
public static void mostrarArray(Object arr[])
{
   for(int i=0;i<arr.length;i++)
   {
      System.out.println(arr[i]);
   }
}
```

ClassCastException 1

Esto sucede si queremos castear un objeto a algo que no es

```
public class Numero
{
   private int valor;

   @Override
   public boolean equals(Object o)
   {
      return this.valor==((Numero)o).valor;
   }

   // ...
}
```

```
String s = “10”;
Numero n = new Numero(10);

if( n.equals(s) )
{
   System.out.prinln(“¿OK?”);
}
```

ClassCastException 2

Esto sucede si queremos castear un objeto a algo que no es

```
@Override
public boolean equals(Object o)
{
   if( !(o instanceof Numero) )
   {
      return false;
   }
   else
   {
      return this.valor==((Numero)o).valor;
   }
}
```

Clase Abstracta 1

¿Qué pasa cuando una clase existe, pero no es algo concreto?

```
public abstract class FiguraGeometrica
{
   public abstract double area();

   public void imprimirArea()
   {
      System.out.println(“Area de la figura:”+area());
   }
}
```

Clase Abstracta 2 (Círculo)

Las siguientes clases “son” figuras geométricas concretas

```
public class Circulo extends FiguraGeometrica
{
   private double radio;

   public Circulo(int r){ this.radio=r; }

   @Override
   public double area()
   {
      return Math.PI*Math.pow(radio,2); // PI*radio al cuadrado
   }
}
```

Clase Abstracta 3 (Rectángulo)

Las siguientes clases “son” figuras geométricas concretas

```
public class Rectangulo extends FiguraGeometrica
{
   private double base,altura;

   public Rectangulo(double b,double h){ this.base=b,this.altura=h; }

   @Override
   public double area()
   {
      return base*altura;
   }
}
```

Clase Abstracta 4 (Triángulo)

Las siguientes clases “son” figuras geométricas concretas

```
public class Triangulo extends FiguraGeometrica
{
   private double base,altura;

   public Triangulo(double b,double h){ this.base=b,this.altura=h; }

   @Override
   public double area()
   {
      return base*altura/2;
   }
}
```

Clase Abstracta 5

Las siguientes clases “son” figuras geométricas concretas

```
public static void areaPromedio(FiguraGeometrica arr[])
{
   double sum=0;
   for(FiguraGeometrica fg:arr)
   {
      sum+=fg.area();
   }

   return sum/arr.length;
}
```

Clase Abstracta 6

Efectivamente, el array arr contenía **figuras geométricas concretas.**

```
FiguraGeometrica arr[] = new FiguraGeometrica[3];
arr[0] = new Circulo(10);
arr[1] = new Rectangulo(5,3);
arr[2] = new Triangulo(4,6);

double ap = areaPromedio(arr);
System.out.println(ap);
```

Excepciones

Constituyen un mecanismo a través del cual **los métodos pueden finalizar abruptamente, arrojando una "excepción".**

Existen dos tipos de excepciones: **Declarativas y No Declarativas**.

Las "declarativas" son subclases de **Exception**, las "no declarativas" son subclases de **RuntimeException**.

La declarativas requieren try-catch, las otras no.

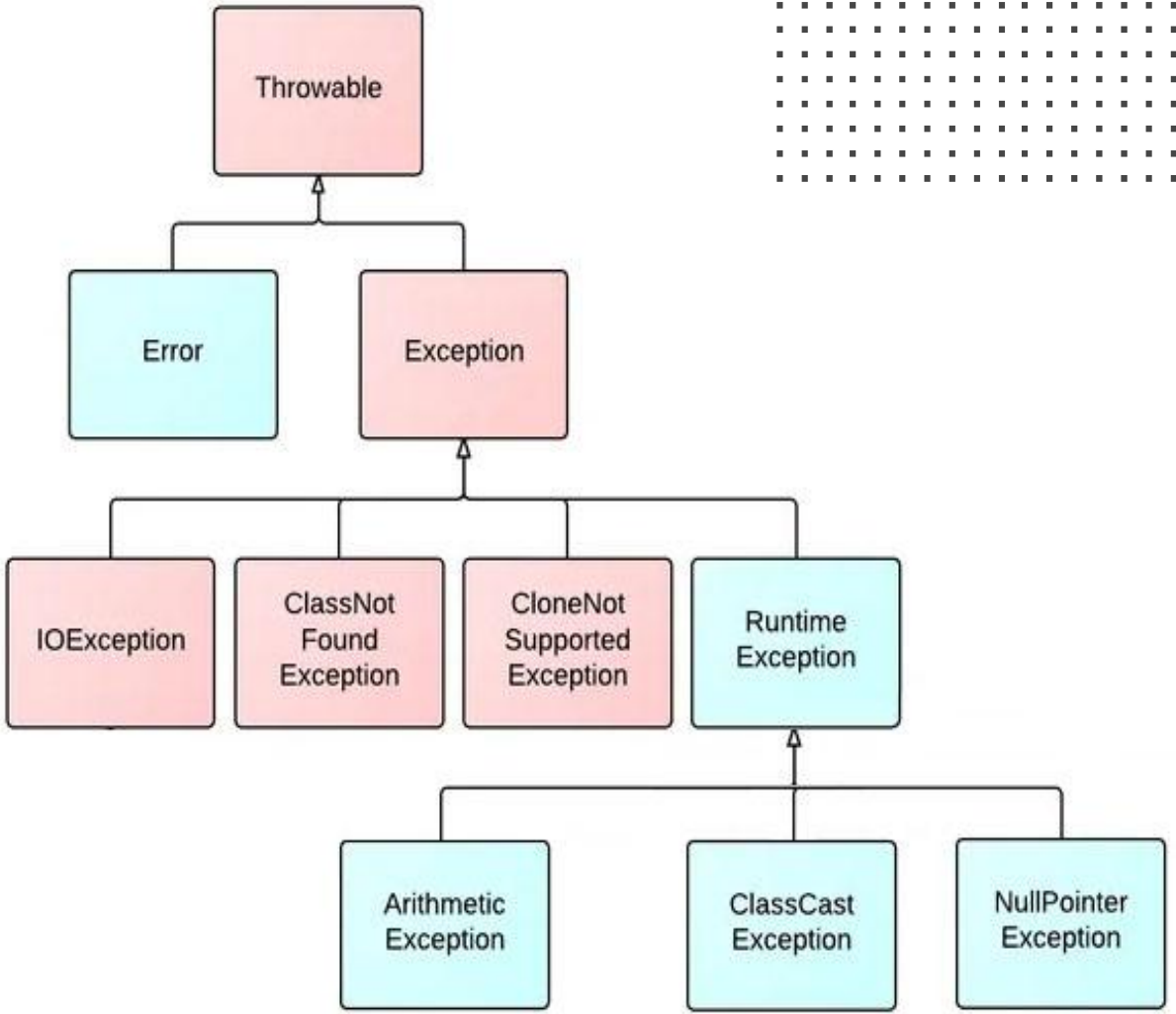
Excepciones 2

Declarativas y No Declarativas.

```
try
{
   FileInputStream fis =
       new FileInputStream(“a.txt”);
   // :
}
catch(FileNotFoundException ex)
{
   String mssg=“No existe el archivo”;
   System.out.println(mssg);
}
```

```
int arr[] = new int[5];
for(int i=0; i<10; i++)
{
   arr[i]=0;
}
```

Excepciones 3



Excepciones 4

La pila de llamadas, el **stack trace**

```
try
{
   FileInputStream fis =
       new FileInputStream("a.txt");
   // :
}
catch(FileNotFoundException ex)
{
   ex.printStackTrace();
   // :
}
```

```
try
{
   FileInputStream fis =
       new FileInputStream("a.txt");
   // :
}
catch(Exception ex)
{
   ex.printStackTrace();
   // :
}
```

Excepciones 5

El bloque completo **try-catch-finally**

```
try{
   System.out.println(“Hola y chau”);
   return;
}
catch(Exception ex){
   ex.printStackTrace();
}
finally {
  System.out.println(“Esto sale siempre”);
}
```

Excepciones 6

Más sobre la sección **finally**

```
try{
   int i = Integer.parseInt(“123 Tira una excepcion 456”);
   System.out.println(i);
}
catch(ArrayIndexOutOfBoundsException ex){
   ex.printStackTrace();
}
finally {
  System.out.println(“Esto sale siempre”);
}
```

Excepciones 7

El bloque completo: **try-catch-finally**

```
FileInputStream fis = null;
try
{
   fis = new FileInputStream(“a.txt”);
   // :
}
catch(FileNotFoundException ex){
   ex.printStackTrace();
}
finally {
  if( fis!=null )fis.close();
}
```

¿Dudas? ¿Preguntas?

Gracias.

IT BOARDING

BOOTCAMP