IPOO

- Docentes
  - Profesores Adjuntos:
    - Carlos Di Cicco.
  - JTPs:
    - Federico Naso (Junín) .
      Nelson Di Grazia (Pergamino) .

Autoboxing / unboxing

- Java es fuertemente tipado
- En Java los tipos pueden ser:
  - Primitivos (minúsculas)
  - Objetos (Clases, mayúsculas)
- Los tipos primitivos tienen un valor predeterminado.
- Existe el ‘null’, valor predeterminado de objetos sin instanciar.

Autoboxing / unboxing

- Wrapper (envoltorio): son clases que modelan los tipos de datos primitivos tales como enteros y flotantes.
- Estos tipos primitivos son los únicos elementos en Java que no son clases.
- Los wrapper ofrecen métodos de conversión muy convenientes:
  - Integer x= new Integer(10) ;
  - x.doubleValue() ;.

Autoboxing / unboxing

- Los Wrappers existentes son:
  - Byte para byte
  - Short para short
  - Integer para int
  - Long para long
  - Boolean para boolean
  - Float para float
  - Double para double
  - Character para char

Autoboxing / unboxing

- Autoboxing: conversión automática de tipo primitivo a su wrapper.
- Unboxing: conversión automática de un wrapper a un tipo primitivo.
- Por ejemplo:
  - Autoboxing: conversión automática de int a Integer.
  - Unboxing: conversión automática de Integer a int.

Autoboxing / unboxing

- Ejemplo sin Autoboxing:
  - Integer y = new Integer(10);
  - int x = y.intValue();
  - x = x + 15;
  - y = new Integer(x);
  - System.out.println("y = " + y);

- Ejemplo con Autoboxing:
  - Integer y = new Integer(10);
  - y = y + 15;
  - System.out.println("y = " + y)

Autoboxing / unboxing

- Una de las mayores utilidades de este es al utilizarlo en Colecciones, por ejemplo:
- Sin Autoboxing:
  - int valor=10;
  - ArrayList values= new ArrayList();
  - values.add(new Integer(valor));
- Con Autoboxing:
  - int valor=10;
  - ArrayList<Integer> values= new ArrayList<Integer>();
  - values.add(valor);

Autoboxing / unboxing

- ¿Cuándo usar autoboxing?
  - Cuando hay una incompatibilidad entre los tipos de referencia y los datos primitivos.
  - Aunque esto no es apropiado para cálculos científicos. Un Integer no es un substituto para un int.
  - El autoboxing y unboxing simplemente ocultan las diferencias entre los wrappers y los datos primitivos.

Autoboxing / unboxing

- if (a == (b+1))
  - Siendo a y b objetos Integer, compara los tipos básicos int. Hace el unboxing de b para sumarlo con 1, y de a para compararlo con el int (b+1).
  - Esta tarea se realiza de forma automática y transparente.

Casting

- En muchas ocasiones hay que transformar una variable de un tipo a otro, por ejemplo de int a double, o de float a long.

- En otras ocasiones la conversión debe hacerse entre objetos de clases diferentes, aunque relacionadas mediante la herencia.

Casting

- Casting entre tipos primitivos
  - La conversión entre tipos primitivos es más sencilla. En Java se realizan de modo automático conversiones implícitas de un tipo a otro de más precisión, por ejemplo de int a long, de float a double, etc.
  - Estas conversiones se hacen al mezclar variables de distintos tipos en expresiones matemáticas o al ejecutar sentencias de asignación en las que el miembro izquierdo tiene un tipo distinto (más amplio) que el resultado de evaluar el miembro derecho.

Casting

- Las conversiones de un tipo de mayor a otro de menor precisión requieren una orden explícita del programador, pues son conversiones inseguras que pueden dar lugar a errores (por ejemplo, para pasar a short un número almacenado como int, hay que estar seguro de que puede ser representado con el número de cifras binarias de short). A estas conversiones explícitas de tipo se les llama cast. El cast se hace poniendo el tipo al que se desea transformar entre paréntesis:
  - long result = (long) (a/(b+c));
  - double flotante = 4; //automático

Casting entre objetos

- Solo puede realizarse entre objetos compatibles, es decir que tengan alguna relación jerárquica:

  Persona (apellido, nombre, dni)

  |_ Alumno (legajo, promedio)

- Si bien el casting siempre es permitido (siempre compila), no siempre es válido (falla en ejecución)

Casting entre objetos

Ejemplos:

```
Persona persona = new Persona();
Alumno alumno = new Alumno();

int legajo = persona.getLegajo(); //no compila

double promedio = ((Alumno) persona).getPromedio();
//compila pero falla en ejecución

String apellido = persona.getApellido(); //OK!
String nombre = alumno.getNombre(); // OK!
int dni = ((Persona) alumno).getDni(); // OK!
```

Fechas

- La clase java.util.Date es la encargada de la administración de Fechas y horas.
  - java.util.Date fecha = new java.util.Date();
- El manejo de fechas se realiza con clases del paquete java.util

Fechas

- Para crear una fecha en particular a través de un String necesitamos utilizar una clase adicional, SimpleDateFormat:

```
SimpleDateFormat sdf = new
    SimpleDateFormat("dd/MM/yyyy");
Date fecha = sdf.parse("30/09/2001");
System.out.println(sdf.format(fecha)); //imprime un String
    con formato
System.out.println(fecha); //imprime la fecha
```

Calendar

- Para manipular fechas se utiliza Calendar:

```
Calendar c = Calendar.getInstance(Locale.getDefault());
c.set(Calendar.DAY_OF_MONTH,3);
c.set(Calendar.MONTH,calendar.JULY);
c.set(Calendar.YEAR,2006);
Date d = c.getTime();
c.add(Calendar.YEAR,2);
d = c.getTime();
```

Calendar – Ejemplo edad

```
public static int getEdad(Date d, Date hasta){

int difference = 0;
   Calendar earlier = Calendar.getInstance();
   Calendar later = Calendar.getInstance();
   earlier.setTime(d);
   later.setTime(hasta);
   difference = later.get(Calendar.YEAR) - earlier.get(Calendar.YEAR);
   if (earlier.get(Calendar.DAY_OF_YEAR) <
     later.get(Calendar.DAY_OF_YEAR))
      difference--;
   return difference;

 }
```

Java.time - Fechas y horas

En Java 8 el paquete java.time es una extensión a las clases java.util.Date y java.util.Calendar que es un poco limitado para manejo de fechas, horas y localización.

Enum de mes y de día de la semana

Existe un enum donde se definen todos los días de la semana.

Lo cual tiene sentido hacerlo enum porque siempre habrán siete días de la semana :).

Este enum se llama java.time.DayOfWeek

```
DayOfWeek lunes = DayOfWeek.MONDAY;
```

Este enum tiene algunos métodos interesantes que permite manipular días hacía adelante y hacia atrás:

```
DayOfWeek lunes = DayOfWeek.MONDAY;
System.out.printf("8 días será: %s%n",lunes.plus(8));
System.out.printf("2 días antes fue: %s%n",lunes.minus(2));
```

Fecha

Las clases de fecha como el java.time.LocalDate manejan la fecha, pero, a diferencia del java.util.Date solo trabaja fecha, y no hora.

Esto permite manipular la fecha de forma específica. Ejemplos:

```
LocalDate date = LocalDate.of(1999, Month.AUGUST, 23);
DayOfWeek dia=date.getDayOfWeek();
System.out.printf("El día que conocí a mi amigo fue el %s y fue un %s%n",date,dia);
```

Mes

Para representar el mes de un año específico, usamos la clase java.time.YearMonth y también podemos obtener la cantidad de días de ese mes.

```
YearMonth mes = YearMonth.now();
    System.out.printf("Este mes es %s y tiene %d días%n", mes, mes.lengthOfMonth());
    mes = YearMonth.of(2004, Month.FEBRUARY);
```

Local time – Clase de hora

La clase java.time.LocalTime se centra únicamente en la hora. Es muy útil para representar horas y tiempos de un día.

Con el java.util.Date solo podemos manipular la hora de un día de un año en especial, de una zona de horario en especial, pero con el LocalTime solo nos centramos en la hora en sí, sin importar que día sea.

Ejemplo de uso:

```
LocalTime justoAhora = LocalTime.now();
System.out.printf("En este momento son las %d horas con %d minutos y %d segundos\n", justoAhora.getHour(),
    justoAhora.getMinute(), justoAhora.getSecond());
```

Local Date Time - Hora/fecha

La clase java.time.LocalDateTime manipula la fecha y la hora sin importar la zona horaria. Esta clase es usada para representar la fecha (año, mes, día) junto con la hora (hora, minuto, segundo, nanosegundo) y es la combinación de LocalDate y LocalTime.

Ejemplos:

```
LocalDateTime ahora = LocalDateTime.now();
System.out.printf("La hora es: %s%n", ahora);
LocalDateTime algunDia = LocalDateTime.of(1976,
    Month.MARCH, 27, 6, 10);
System.out.printf("Yo nací el %s%n", algunDia);
System.out.printf("Hace seis meses fue %s%n",
    LocalDateTime.now().minusMonths(6));
```

Enumerativos - Enum

- Un enumerated type (Enum) es un tipo de dato que contiene un conjunto de valores constantes.
- Las características avanzadas de la OO incluyen la capacidad de agregar métodos y campos a los enums.
  - enum Season {WINTER, SPRING, SUMMER, FALL}
  - enum TamanoDeCafe{CHICO,MEDIANO,GRANDE}
  - enum Day {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY};

Enumerativos - Enum

- Los Enums son Comparable y Serializables.
- Enums son básicamente un nuevo tipo de Clase.
- ¿Cuando debería usar Enums?
  - Tipos de enumerados naturales: días de la semana, fases de la luna, estaciones.
  - Otros conjuntos donde sepamos todos los valores posibles: opciones de un menú, etc.
  - Como reemplazo de los flags (EnumSet)

Enumerativos - Enum

- Debido a que los tipos enumerados son como una clase de tipo especial en Java, hay muchas cosas que se pueden realizar dentro de un enum.
- Además de declarar constantes, un tipo enumerado puede contener constructores, métodos y variables.

Enumerativos - Ejemplo

```
enum TamanoCafe{
    CHICO(5), MEDIANO(8), GRANDE(10);
private int onzas;
TamanoCafe(int onzas){
    this.onzas = onzas;
}
public int getOnzas(){
    return this.onzas;
}
}
…
TamanoDeCafe tdc =
    TamanoDeCafe.MEDIANO;
tdc.getOnzas(); -> 8
```

Enumerativos

- Los Enum se pueden iterar:
  - for (TamanoDeCafe p : TamanoDeCafe.values()){
  - …
  - }
- Se puede pedir el valor:
  - TamanoDeCafe.valueOf("MEDIANO")

String

- Los Strings son una secuencia de caracteres.
- Java no posee un tipo de dato primitivo para las cadenas de caracteres.
- En lugar de ello, la clase String sirve para crear y almacenar una secuencia de caracteres como objetos String.
- La clase String se define como final. Esto significa que cualquier objeto creado a partir de esta clase se considera como inmutable, esto es, no puede cambiar.

String

- Para definir Strings, las cadenas de caracteres se deben encerrar entre comillas dobles y pueden crearse con **el operador new o como literal:**
  - String perro= new String(“Scooby Do”);
  - String mascota = “Pumba”;
  - return “Hola”; // se pueden crear sin referencia, en un método
  - System.out.println(“Bienvenidos!!”); //crear como parámetro

String

- Al crear una variable que haga referencia a un objeto String, un objeto de tipo String diferente puede asignarse a la variable de referencia
  - String mascota = “Timon”;
  - mascota= new String(“Pumba”); // el String “Timon” ya no tiene referencia.
  - mascota= mascota+ “y Timon”; // el String“Pumba” ya no tiene referencia

String

- String str1 = "Hola";
- String str2 = new String("Hola");
- El primero de los métodos expuestos es el más eficiente, porque como al encontrar un texto entre comillas se crea automáticamente un objeto String, en la práctica utilizando new se llama al constructor dos veces.

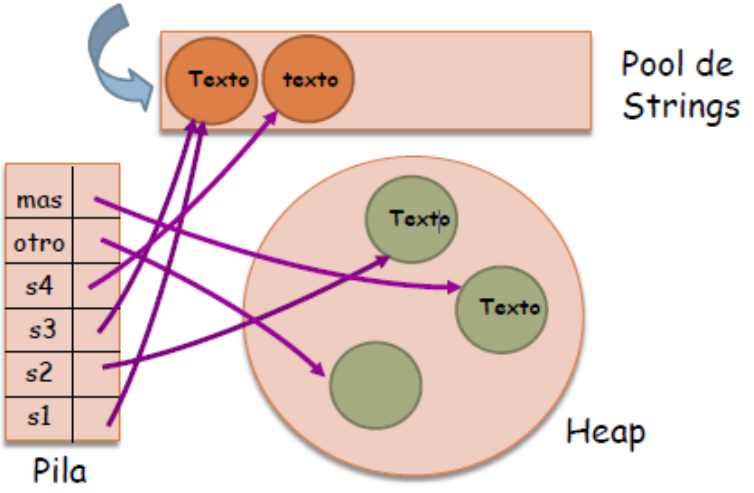
String

- A pesar que los Strings son inmutables, pueden compartirse, cuando se crea una cadena utilizando una asignación literal, la cadena se almacena en un espacio de memoria que puede ser compartido por los objetos String. Esto a menudo se denomina Pool de Strings.

```
// Se crean Strings para datos fijos

String s1,s2,s3,s4;
s1 = "Texto";
s2 = new String("Texto");
s3 = "Texto";
s4 = "texto";

String otro = new String();
String mas = new String(" Texto");
```



String

- La clase ***String*** incluye métodos para examinar caracteres individuales de una secuencia, para comparar strings, para buscar substrings, para extraer substrings, para convertir todo el string a mayúsculas o minúsculas, etc.
  - String s = new String("Jirafa");
  - s.**substring(1,3)**; // ir
  - s.**substring(0,3)**; // Jir
  - s.**charAt(2)**; // r
  - s.**toUpperCase()**; // JIRAFA

String

- La igualdad de cadenas utilizando el operador == sólo probará la igualdad de las referencias. Para llevar a cabo una comparación en mayor profundidad de los datos de los objetos, utilice un método compareTo() o equals() del objeto.
  - String s = new String(“Jirafa“);
  - String s1 = new String(“Jirafa”);
  - s.**equals(new String("Jirafa"));**//true
  - s**==**s1; // false
  - s.**compareTo(new String("Jirafa"));**// 0

String

- Además de estos métodos, el lenguaje Java provee soporte especial para la ***concatenación***(+): El operador + se considera un operador sobrecargado en Java. Como operandos, acepta valores primitivos numéricos y valores de tipo ***String***.
  - String s = “Hola”;
  - s = s + “ que tal”;
  - s = s + 2;

String y StringBuffer

- Las clases String y StringBuffer están orientadas a manejar cadenas de caracteres.
  - String está orientada a manejar cadenas de caracteres constantes, es decir, que no pueden cambiar.
  - StringBuffer permite que el programador cambie la cadena insertando, borrando, etc.
- La primera es más eficiente, mientras que la segunda permite más posibilidades.
- Ambas clases pertenecen al paquete java.lang, y por lo tanto no hay que importarlas.
- El operador de concatenación (+) entre objetos de tipo String utiliza internamente objetos de la clase StringBuffer y el método append().

String y StringBuffer

- String y StringBuffer son clases independientes. No poseen métodos y campos comunes.
- StringBuffer se debe usar para manipular los caracteres y la clase String como el lugar donde se coloca el resultado final.
- Cada objeto StringBuffer tiene una capacidad o tamaño máximo. No obstante, si se agregan caracteres adicionales a un StringBuffer de modo tal que se exceda su capacidad, Java proporcionará una capacidad mayor.

StringBuffer

- Ejemplos:
  - StringBuffer = new StringBuffer(“¿Cómo estás”);
  - s.append(“?”); //¿Cómo estás?
  - s.chaAt(2); //ó
  - s.delete(7,10); // ¿Cómo es?
  - s.insert(8," el niño");//¿Cómo es el niño?
  - s.reverse(); //?oñin le se omóC¿

Paquetes

- En Java, una librería de componentes (clases e interfaces) es un grupo de archivos .class, también llamada paquete.
- Para agrupar componentes en una librería o paquete, debemos anteponer la palabra clave package al comienzo del archivo fuente de cada una de las componentes.
- Convención de nombres: minúscula para nombres de paquetes e inicial mayúscula para nombres de clases.

Paquetes

```
package graficos;
//Establece que la clase Rectangle pertenece al paquete graficos
    public class Rectangle extends Graphics {
    . . .
    }
```

- Cualquiera que quiera usar la clase Rectangle debe especificar el nombre completo o usar la palabra clave import:

```
import graficos.Rectangle;
import graficos.*;
//Están disponibles todas las clases public en el paquete graficos
Rectangle r=new Rectangle();
graficos.Rectangle r=new graficos. Rectangle();
 //Nombre completo de la clase Rectangle
```

Paquetes – Nombres únicos

- Los únicos paquetes que se importan automáticamente (no requieren usar la sentencia import) son el paquete java.lang y el paquete actual (el que estamos trabajando).
- Las clases e interfaces que se definen sin usar la sentencia package, se ubican en el paquete de default (default package). No se recomienda usar el default package.
- En Java, los nombres de los miembros de una clase (atributos y nombres) están aislados: el método get() de la clase A no colisiona con el método get() de la clase B.

Paquetes – Nombres únicos

- Pero, ¿ qué sucede con los nombres de clases? ¿Cómo evitar el conflicto de nombres?
- Los paquetes nos permiten dividir el espacio de nombres global y así evitar colisión de nombres de clases: supongamos que escribimos la clase Pila y la instalamos en una máquina que ya tiene la clase Pila. Aquí se plantea un conflicto de nombres. En Java es importante tener control sobre el espacio de nombres; debemos crear nombres únicos.
- Se recomienda usar como primera parte del nombre del paquete, el nombre invertido del dominio de Internet o crear una combinación de apellido y nombres, de esta manera estamos garantizando que el nombre del paquete es único y de esta manera evitamos conflictos con los nombres de clases.

Paquetes – Nombres únicos

- ¿Qué sucede si se importan dos librerías que incluyen el mismo nombre de clase?

```
import graficos.*;
import java.awt.*;
//Ambas contienen la clase Rectangle
Rectangle r=new Rectangle();
//El compilador no puede determinar a que clase referencia
java.awt.Rectangle r=new java.awt.Rectangle(); //Escribir el nombre completo de la clase
```

Paquetes – Ubicación de archivos

- Los paquetes no necesariamente están "empaquetados" en un único archivo, esto es, todos los .class que componen un paquete no siempre están ubicados juntos en un mismo archivo.
- Java saca provecho de la estructura jerárquica de directorios del SO y ubica todos los .class de un mismo paquete en un directorio. De esta manera, se resuelve:
  - El nombre único del paquete
  - La búsqueda de los .class (que de otra forma estarían diseminados en el disco)

Paquetes – Ubicación de archivos

- Los nombres de paquetes se resuelven en directorios del SO. En el nombre del paquete se codifica el path de la ubicación de los .class:

```
package graficos;
public class Rectangle extends Graphics {
. . .
}
```

El archivo se encuentra en graficos/Rectangle.class

- Cuando el ínterprete Java, ejecuta un programa y necesita localizar dinámicamente un archivo .class (cuando se crea un objeto o se accede a una variable static), procede de la siguiente manera:
  - Busca en los directorios estándares (donde está instalado el JRE y en el actual)
  - Recupera la variable de entorno CLASSPATH, que contiene la lista de directorios usados como raíces para buscar los archivos .class.
  - Toma el nombre del paquete, de las sentencias import, y reemplaza cada "." por una barra "\" 0 "/" (según el SO) para generar un path a partir de las entradas del CLASSPATH.

Paquetes – Archivos JAR

- Es posible agrupar todos los .class de un paquete en un único archivo: Usando archivos JAR (Java ARchive). En ese caso, en el CLASSPATH, se especifica el nombre del archivo JAR
- El intérprete Java se encarga de buscar, descompactar, cargar e interpretar estos archivos.
- El formato JAR usa el formato Zip. Los archivos JAR son multiplataforma. Es posible incluir además de archivos .class, archivos de imágenes y audio, etc.
- El JSE o JDK de Sun tiene una herramienta que permite crear archivos JAR, desde la línea de comando, es el utilitario jar.

Paquetes – Ejemplos

```
package ar.edu.unnoba.ipoo.graficos;
public class Externa {
.......
}
class Interna{
.....
}
```

- Existe un único archivo fuente: Externa.java
- El compilador genera dos archivos .class:
  - \ar\edu\unnoba\metodologias\graficos\Externa.class
  - \ar\edu\unnoba\metodologias\graficos\Interna.class

Paquetes – Ejemplos

- El archivo fuente Java, se llama unidad de compilación y tiene extensión .java.
- El nombre del archivo fuente Java debe coincidir, incluyendo mayúsculas y minúsculas, con el nombre de la clase o interface declarada public.
- En el archivo fuente Java solo se declara una clase o interface public y si hay clases o interfaces adicionales, éstas deben declararse no-public o package.
- El compilador crea diferentes archivos de salida, uno para cada una de las clases o interfaces contenidas en el archivo fuente. Cada uno de ellos, tiene el nombre de la clase o interface y extensión .class

Paquetes – Especificación de acceso

- Permiten al autor de una librería de componentes establecer qué está disponible para el usuario (cliente programador) y qué no.
- Los niveles de control de acceso son:
  - public (Más libre)
  - protected
  - package (no tiene palabra clave)
  - private (Más restrictivo)
- El control de acceso permite ocultar la implementación. Separa la interfaz de la implementación, permite hacer cambios que no afectan al código del usuario de la librería.

Paquetes – Especificación de acceso

- En Java, los especificadores de acceso se ubican adelante de la definición de cada atributo o método de una clase. El especificador, solamente controla el acceso dicha definición
- ¿Qué pasa si a un miembro de una clase no le definimos especificador de acceso?
- Tiene acceso por defecto, no tiene palabra clave y comúnmente se lo llama acceso package o friendly . Implica que tienen acceso a dicho miembro, todas las clases ubicadas en el mismo paquete que él y, para las restantes clases es un miembro privado.
- El acceso package le da sentido a agrupar clases en un paquete.

Paquetes – Especificación de acceso

- El atributo o método declarado public está disponible para TODOS. Es lo que usuario "ve".
- El atributo o método declarado private solamente está accesible para la clase que lo contiene. Está disponible para usar adentro de los métodos de dicha clase. Una buena práctica es declarar private todo lo posible.

Paquetes – Especificación de acceso

- La palabra protected, está relacionada con la herencia:
  - Si la subclase se crea en un paquete diferente que el de la superclase o clase base, a los únicos miembros de la clase base que se tienen acceso es a los definidos public.
  - Si la subclase pertenece al mismo paquete que la superclase o clase base, se tiene acceso a todos los miembros declarados public y package.
  - El autor de la clase base, podría determinar que ciertos miembros pueden ser accedidos por las clases derivadas, pero no por todo el mundo. Esto es protected.
  - Además, el acceso protected provee acceso package: las clases declaradas en el mismo paquete que el miembro protected, tienen acceso a dicho miembro.

Paquetes – Especificación de acceso

- En Java, los especificadores de acceso en clases, se usan para determinar cuáles son las clases disponibles de una librería, para los usuarios programadores.
- Una clase declarada public, está disponible para los programadores, mediante la cláusula import. Se pueden crear instancias de la clase.

```
package prueba;
public class Test{
 .....
}
```

Paquetes – Especificación de acceso

- ¿Qué pasa si en el paquete prueba, tengo una clase que se usa de soporte para la clase Test y para otras clases públicas del paquete prueba?
- La definimos de acceso package y de esta manera, solamente puede usarse en el paquete prueba. Es razonable que los miembros de una clase de acceso package tengan también acceso package.

```
package prueba;
class Soporte{
// La clase Soporte puede usarse solamente en el paquete
    prueba
}
```

Paquetes – Sobrescritura de métodos

- Cualquier método que se herede (no privado y no final) de una superclase puede ser sobreescrito en las subclases.
- Los métodos sobreescritos en una subclase, deben tener el mismo nombre, la misma lista de argumentos (en cuanto a tipo y orden) y el mismo tipo de retorno que los de la clase padre.
- El nivel de acceso de un método sobreescrito debe ser igual o menos restrictivo que el declarado en la superclase. Por ejemplo: si en la superclase, el método es declarado public, entonces el método sobreescrito en la subclase debe declararse public. Si en la superclase el método es declarado protected o default, en la subclase puede declararse public.

Paquetes – Sobrescritura de métodos

- Las Excepciones, son clases especializadas que representan errores que pueden ocurrir durante la ejecución de un método. Los métodos sobreescritos deben disparar las mismas excepciones o subclases de las excepciones disparadas por el método de la superclase. No pueden disparar otras excepciones.