Desarrollo Avanzado de Aplicaciones I

Excepciones

idat

i d a t

inicio desarrollo aplicación término

idat

Inicio

Logro de aprendizaje - Introducción

idat

Logro de Aprendizaje

“Al finalizar la sesión, el participante podrá proteger sus aplicaciones con el uso de excepciones, crear nuevas excepciones y capturar y gestionar las excepciones a través de la sentencia try...catch...finally, Adicionalmente, podrá usar arreglos y colecciones.

idat

Introducción

- Revisión Rápida de Temas de Sesión Anterior
- Revisión de Ejercicios de Sesión Anterior
- Inquietudes y/o Preguntas
- Agenda de Sesión
  - Arreglos
  - Colecciones
  - Gestión de Excepciones
  - Sentencia try … catch … finally
  - Creación de Excepciones
  - Ejemplos y Ejercicios

Desarrollo

Desarrollo del Contenido de la Sesión

idat

Arreglos

- ✓ Un arreglo es una estructura de datos (colección) utilizada para almacenar datos del mismo tipo. Los arreglos almacenan sus elementos en ubicaciones de memoria contiguas.
- ✓ Existen 2 tipos de arreglos, los unidimensionales (también denominados vector) y los multidimensionales (también denominados matrices).
- ✓ En Java, los arreglos son objetos, por lo tanto heredan de la clase "Object" sus principales métodos. Y también son estáticos, es decir el tamaño de los arreglos se declara en un primer momento y no puede cambiar luego durante la ejecución de un programa.
- ✓ Los arreglos se numeran desde el elemento cero, que sería el primer elemento, hasta la capacidad-1 que sería el último elemento. Es decir, si tenemos un arreglo de diez elementos, el primer elemento sería el cero y el último elemento sería el nueve.

Sintaxis de Arreglos Unidimensionales

idat

- ✓ Declaración y creación:
  - ➢ TipoDato[ ] nombreArreglo = new TipoDato[dimension];
- ✓ También puede ser declarado y creado de forma separada:
  - ➢ TipoDato[ ] nombreArreglo;
  - ➢ nombreArreglo = new TipoDato[dimension];
- ✓ Para acceder a un elemento especifico utilizaremos los corchetes de la siguiente forma. Entendemos por acceso, tanto el intentar leer el elemento, como asignarle un valor:
  - ➢ nombreArreglo[numero_elemento]

Sintaxis de Arreglos Multidimensionales

idat

- ✓ Declaración y creación:
  - ➢ TipoDato[ ][ ] nombreArreglo = new TipoDato[dimension][dimension];
- ✓ También puede ser declarado y creado de forma separada:
  - ➢ TipoDato[ ][ ] nombreArreglo;
  - ➢ nombreArreglo = new TipoDato[dimension][dimension];
- ✓ Para acceder a un elemento especifico utilizaremos los corchetes de la siguiente forma. Entendemos por acceso, tanto el intentar leer el elemento, como asignarle un valor:
  - ➢ nombreArreglo[numero_elemento][numero_elemento]

Inicialización de Arreglos

idat

- ✓ Arreglo unidimensional:
  - ➢ TipoDato nombreArreglo[] = { elemento1 , elemento2 , … , elementoN };
- ✓ Arreglo multidimensional:
  - ➢ TipoDato nombreArreglo[][] = { { elemento1 , elemento2 , … , elementoN } , { elemento1 , elemento2 , … , elementoN } };
- ✓ Cuando se realiza una declaración e inicialización no es necesaria la creación.
- ✓ Cuando se crea un arreglo y no se inicializa, sus elementos reciben los valores por defecto (0, vacio y false).
- ✓ Los arreglos tienen el atributo “lenght” para saber su dimensión o tamaño.

Ejemplo de Arreglo Unidimensional

idat

```
int[] numeros = new int[3];
numeros[0] = 21;
numeros[1] = 67;
numeros[2] = 19;
for(int i=0;i<numeros.length;i++)
    System.out.println(numeros[i]);
```

```
21
67
19
```

```
int[] numeros = {21, 67, 19};
for(int i=0;i<numeros.length;i++)
    System.out.println(numeros[i]);
```

Ejemplo de Arreglo Multidimensional

idat

```
int[][] matriz = new int[3][2];
matriz[0][0] = 21;
matriz[0][1] = 67;
matriz[1][0] = 19;
matriz[1][1] = 12;
matriz[2][0] = 76;
matriz[2][1] = 91;
```

```
int[][] matriz = { {21, 12}, {67, 76}, {19, 91}};
```

```
for(int i=0;i<matriz.length;i++) {
    for(int j=0;j<matriz[i].length;j++)
        System.out.print(matriz[i][j] + " ");
    System.out.print( s: "\n");
}
```

```
21 12
67 76
19 91
```

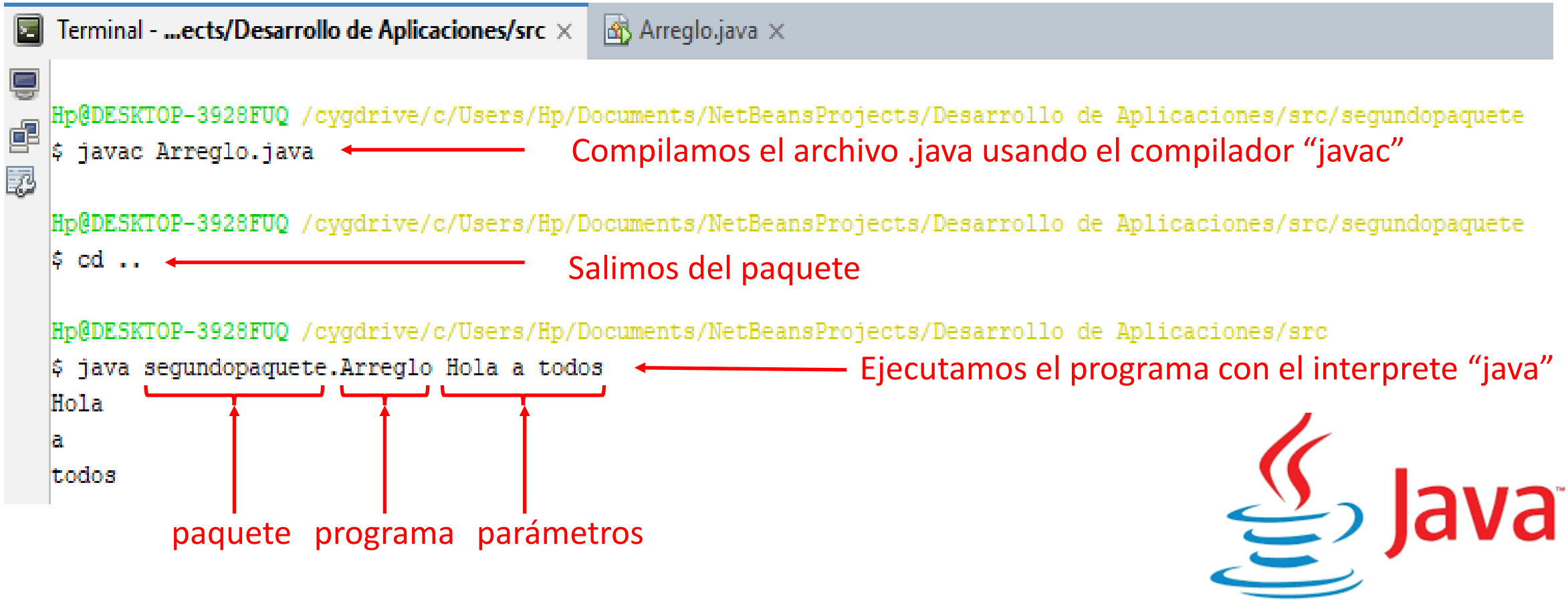
El Arreglo del Método "main"

- ✓ El método "main" recibe como parámetros un arreglo de cadena de caracteres, los cuales se pueden usar en su bloque de código.

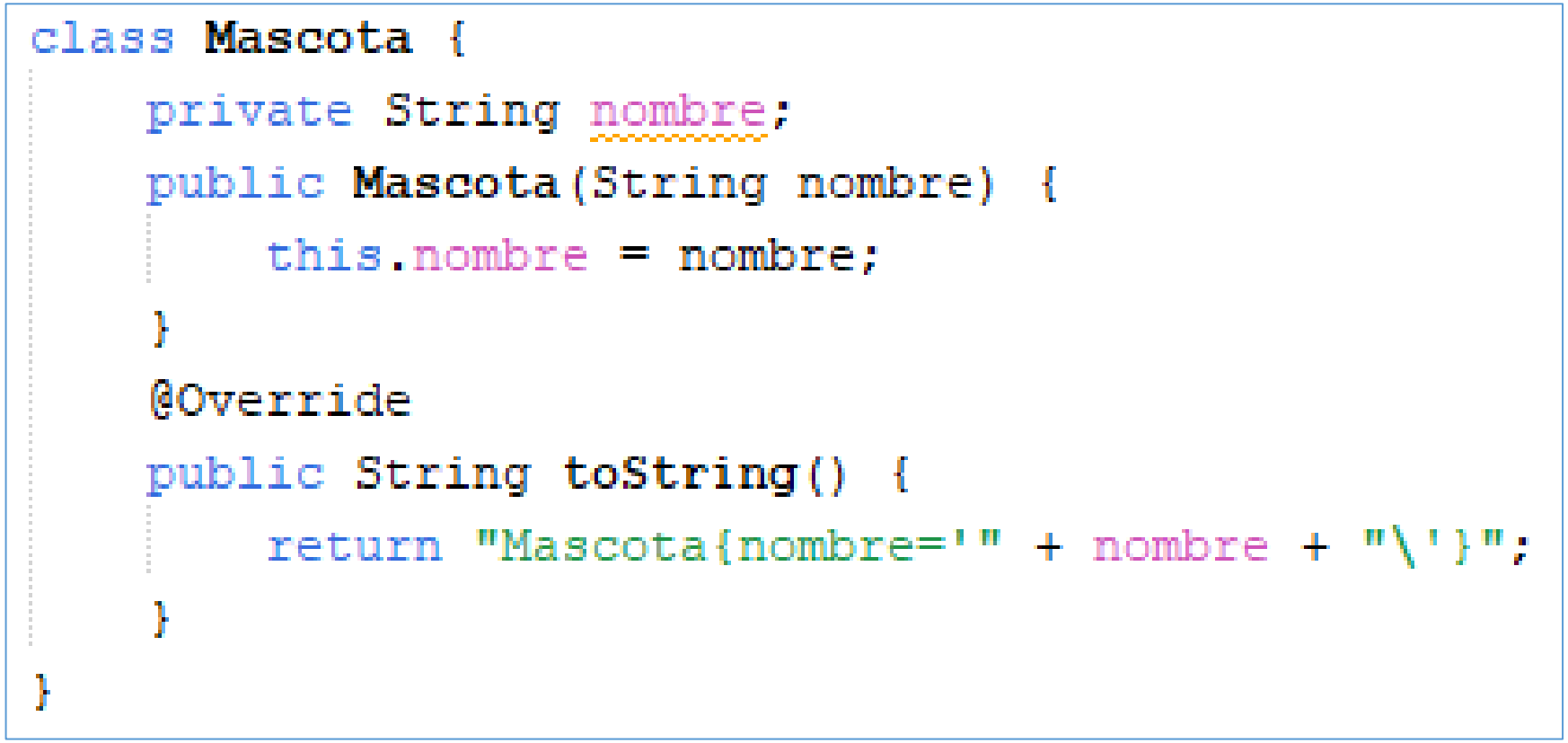
```
public class Arreglo {
    public static void main(String[] args) {
        for(int i=0;i<args.length;i++)
            System.out.println(args[i]);
    }
}
```

- ✓ Para ejecutar el método "main" por consola usando argumentos se usa el siguiente comando:
  - ➢ $ java NombreClase arg1 arg2 arg3 …

El Arreglo del Método main (cont.)



Arreglo de Clases u Objetos



```
class Mascota {
    private String nombre;
    public Mascota(String nombre) {
        this.nombre = nombre;
    }
    @Override
    public String toString() {
        return "Mascota{nombre='" + nombre + "\'}";
    }
}
```

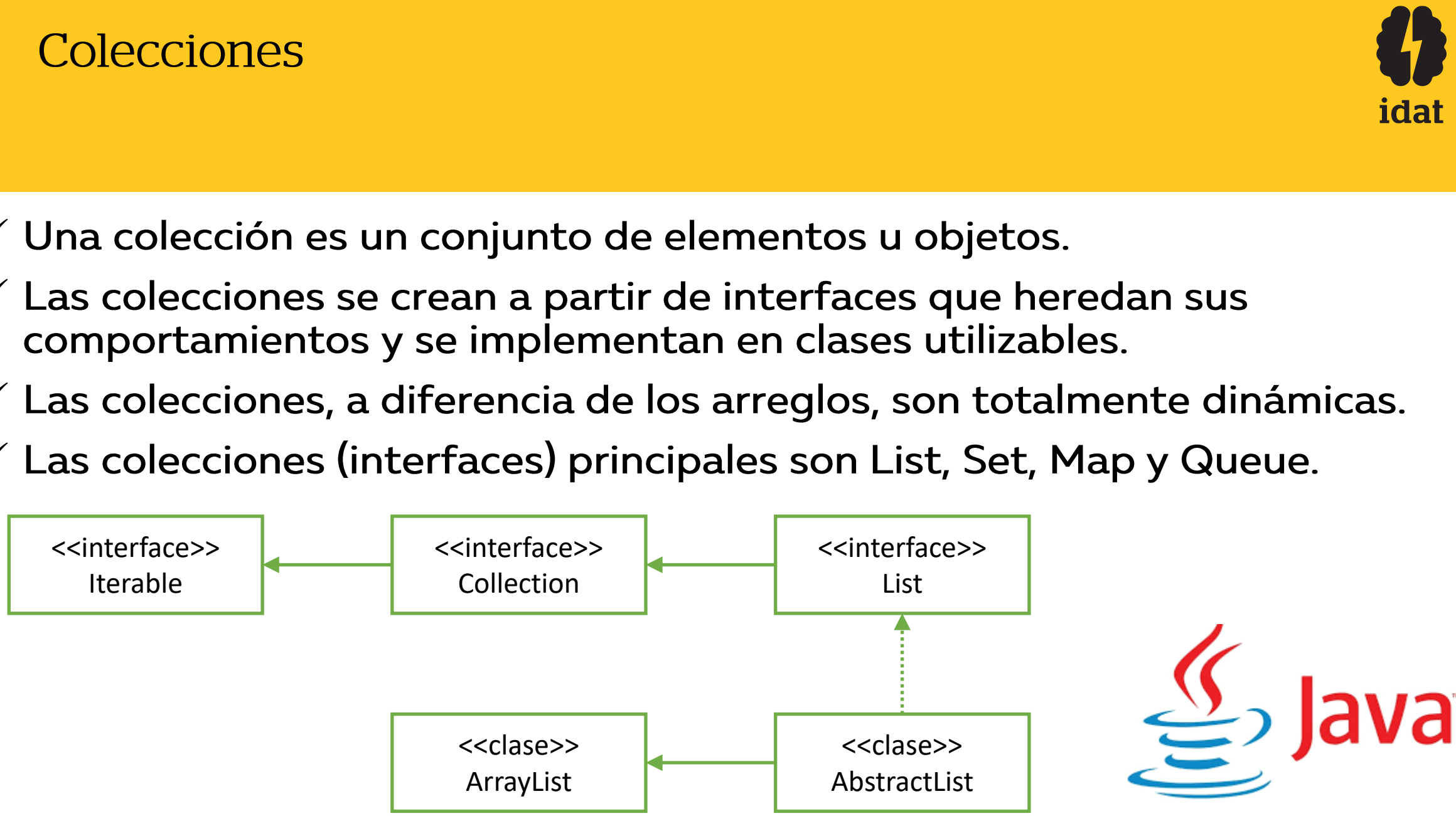
Arreglo de Clases u Objetos (cont.)

idat

```
public class Arreglo {
    public static void main(String[] args) {
        Mascota[] mascotas = new Mascota[2];
        mascotas[0] = new Mascota( nombre: "Dora");
        mascotas[1] = new Mascota( nombre: "Canela");
        for(int i=0;i<mascotas.length;i++)
            System.out.printf( format: "Mascotas[%d] = %s\n",  args: i, mascotas[i]);
    }
}
```

```
Mascotas[0] = Mascota{nombre='Dora'}
Mascotas[1] = Mascota{nombre='Canela'}
```

Colecciones



- ✓ Una colección es un conjunto de elementos u objetos.
- ✓ Las colecciones se crean a partir de interfaces que heredan sus comportamientos y se implementan en clases utilizables.
- ✓ Las colecciones, a diferencia de los arreglos, son totalmente dinámicas.
- ✓ Las colecciones (interfaces) principales son List, Set, Map y Queue.

La Clase ArrayList

idat

- ✓ Un ArrayList es una clase contenedora genérica que implementa arreglos dinámicos de objetos de cualquier tipo.
- ✓ No se pueden usar tipos de datos primitivos. Para un tipo de dato primitivo se debe utilizar su clase envolvente o wrapper.
- ✓ Sintaxis del ArrayList:
  - ➢ ArrayList nombreArrayList = new ArrayList();
  - ➢ ArrayList<tipoClase> nombreArrayList = new ArrayList<tipoClase>();
- ✓ Para utilizar el ArrayList se debe importar la clase que se encuentra en el paquete java.util:
  - ➢ import java.util.ArrayList;

Métodos de la Clase ArrayList

idat

Método	Función
size()	Devuelve el número de elementos (int).
add(X)	Añade el objeto X al final. Devuelve true.
add(posición,X)	Inserta el objeto X en la posición indicada.
get(posicion)	Devuelve el elemento que está en la posición indicada.
clear()	Elimina todos los elementos.
remove(posicion)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.

Método	Función
remove(X)	Elimina la primera ocurrencia del objeto X. Devuelve true si el elemento está en la lista.
set(posición,X)	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
contains(X)	Comprueba si la colección contiene al objeto X. Devuelve true o false.
indexOf(X)	Devuelve la posición del objeto X. Si no existe devuelve -1.

Ejemplo de ArrayList

idat

```
ArrayList<String> nombres = new ArrayList();
nombres.add( e: "Ana");
nombres.add( e: "Luisa");
nombres.add( e: "Felipe");
System.out.println( x: nombres); // [Ana, Luisa, Felipe]
nombres.add( index: 1,  element: "Pablo");
System.out.println( x: nombres); // [Ana, Pablo, Luisa, Felipe]
nombres.remove( index: 0);
System.out.println( x: nombres); // [Pablo, Luisa, Felipe]
nombres.set( index: 0, element: "Alfonso");
System.out.println( x: nombres); // [Alfonso, Luisa, Felipe]
String s = nombres.get( index: 1);
String ultimo = nombres.get(nombres.size() - 1);
System.out.println(s + " " + ultimo);  // Luisa Felipe
```

Excepciones

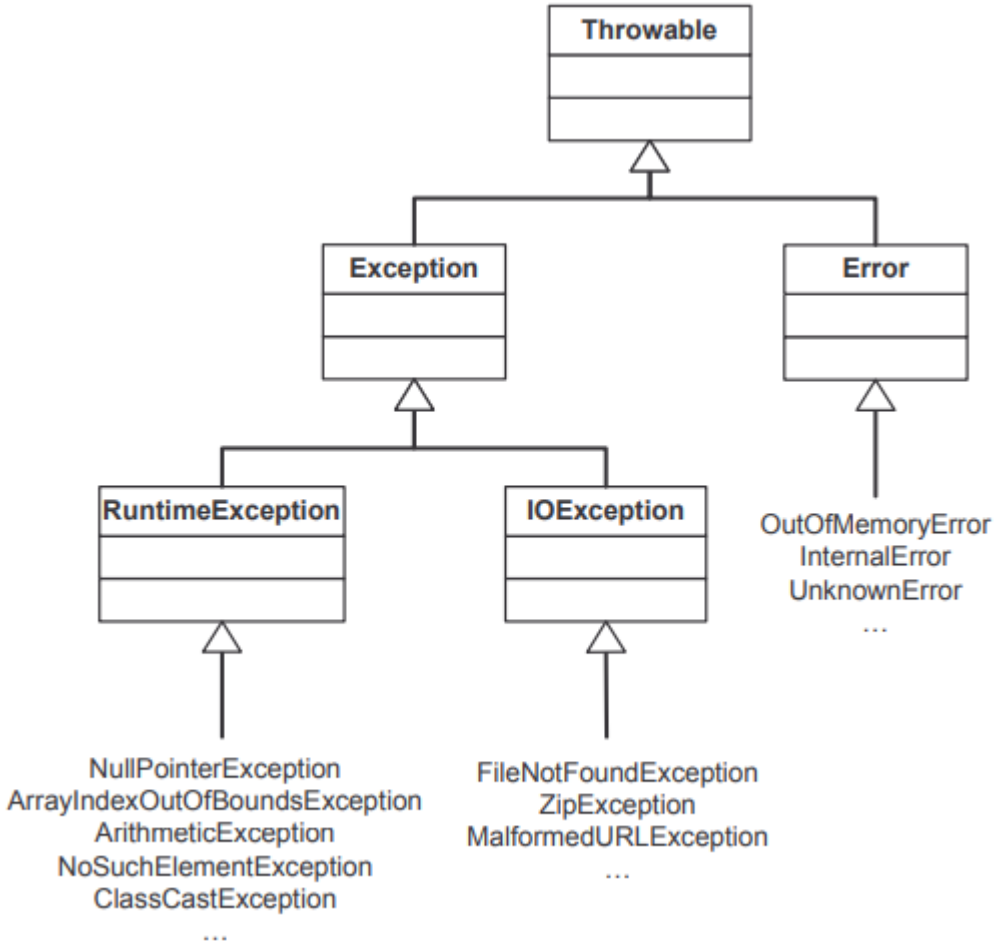
idat

- ✓ Una excepción es un evento que ocurre durante la ejecución de un programa y que interrumpe el flujo normal de las sentencias. Este evento nos indica que una situación excepcional ha ocurrido.
- ✓ Estas situaciones excepcionales son manejadas por el gestor de excepciones de Java, lo cual permite que el código que gestiona las excepciones se encuentre claramente separado del código que las genera.
- ✓ Ejemplos de situaciones excepcionales:
  - ➢ Una avería del disco duro.
  - ➢ Se acabó la memoria.
  - ➢ Dividir por cero.
  - ➢ Acceder a un elemento de un arreglo fuera de sus límites.

Jerarquía de Clases de Excepciones

idat

- ✓ Cuando en Java se produce una excepción se crea un objeto de una determinada clase (dependiendo del tipo de error que se haya producido), que mantendrá la información sobre el error producido y nos proporcionará los métodos necesarios para obtener dicha información.
- ✓ Estas clases tienen como clase padre la clase "Throwable", por tanto se mantiene una jerarquía en las excepciones.



Gestión de las Excepciones

- ✓ Cuando se produce una excepción se crea un objeto de una determinada clase de la jerarquía de clases de las excepciones.
- ✓ Cualquiera de los métodos invocados previamente y que originaron la llamada al método donde se produjo la excepción (llamada pila de llamadas o pila de memoria o stack trace o call chain), puede capturar la excepción y ejecutar las instrucciones convenientes o pertinentes.
- ✓ Tras capturar la excepción, el control no vuelve al método en el que se produjo la excepción, sino que la ejecución del programa continúa en el punto donde se haya capturado la excepción.
- ✓ Las excepciones pueden ser lanzadas por la JVM, aquellas que no es posible que el compilador pueda detectarlas antes del tiempo de ejecución. Y también pueden ser programáticas o lanzadas por el desarrollador de forma explícita a través de la palabra reservada “throw”.

Ejemplo de Excepciones

```java
int[] numeros = new int[3];
numeros[0] = 21;
numeros[1] = 67;
numeros[2] = 19;
numeros[3] = 33;
for(int i=0;i<numeros.length;i++)
    System.out.println(numeros[i]);
```

Error en tiempo de ejecución.
El índice máximo del Arreglo es 2.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
        at segundopaquete.Arreglo.main(Arreglo.java:13)
```

Ejemplo de Excepciones (cont.)


```
int[] numeros = new int[3];
numeros[0] = 21;
numeros[1] = 67;
numeros[2] = 19;
for(int i=0;i<numeros.length;i++)
    System.out.println(numeros[i]/i);
```

Error en tiempo de ejecución.
División por cero.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at segundopaquete.Arreglo.main(Arreglo.java:14)
```

Captura de las Excepciones

- ✓ Para capturar las excepciones en un bloque de código se utiliza la sentencia try … catch.
- ✓ Sintaxis de la sentencia try … catch.

```
try {
        //Bloque de código que puede generar una excepción.
} catch (NombreClaseExcepcion nombreObjeto) {
        //Bloque de código que se ejecuta cuando se produce una
        //excepción. Captura la excepción a través de un objeto de la
        //jerarquía de clases de excepciones. Puede haber
        //mas de una sección catch.
}
```

Captura de las Excepciones (cont.)

- ✓ Se puede agregar la sección “finally” para completar o asegurarse de realizar alguna acción interrumpida o no por una excepción:
  - ➢ try {
  - ➢ //Bloque de código que puede generar una excepción.
  - ➢ } catch (NombreClaseExcepcion nombreObjeto){
  - ➢ //Bloque de código que se ejecuta cuando se produce una
  - ➢ //excepción.
  - ➢ } finally {
  - ➢ //Bloque de código que se ejecuta se produzca o no
  - ➢ //una excepción. Esta sección es opcional.
  - ➢ }

Ejemplo de Captura de Excepciones

```
try {
    int[] numeros = {21, 67, 19};
    for(int i=0;i<numeros.length;i++)
        System.out.println(numeros[i]/i);
} catch (Exception e){
    System.out.println( x: e.toString());
}
```

```
java.lang.ArithmeticException: / by zero
```

Clase general Exception para todas las Excepciones.

Método toString() sobrescrito de la Clase Object, que muestra la clase de la excepción y el error.

Ejemplo de Captura de Excepciones (cont.)

```
try {
    int[] numeros = {21, 67, 19};
    for(int i=0;i<numeros.length;i++)
        System.out.println(numeros[i]/i);
} catch (ArithmeticException ae) {
    System.out.println( x: ae.toString());
}
```

Clase específica ArithmeticException para las excepciones aritméticas.

```
java.lang.ArithmeticException: / by zero
```

Ejemplo de Captura de Excepciones (cont.)

idat

```
try {
    int[] numeros = {21, 67, 19};
    for(int i=0;i<numeros.length;i++)
        System.out.println(numeros[i]/i);
} catch (ArithmeticException ae) {
    System.out.println( x: ae.toString());
} finally {
    System.out.println( x: "La aplicación terminó correctamente");
}
```

```
java.lang.ArithmeticException: / by zero
La aplicación terminó correctamente
```

Declaración de Nuevas Excepciones

- ✓ Podemos crear nuevas excepciones, declarando una nueva clase que herede de la clase "Exception". Esta nueva clase no necesita declarar nuevos atributos ni métodos, solo sus constructores:

```
class IngresoIncorrecto extends Exception {
    public IngresoIncorrecto() {
        super();
    }
    public IngresoIncorrecto(String message) {
        super(message);
    }
}
```

Clase que hereda de la clase Exception

Métodos constructores

Declaración de Nuevas Excepciones (cont.)

- ✓ Para usar la nueva excepción debemos declararla en el método donde se lanzará la excepción, a través de la cláusula "throws".
- ✓ Para lanzar la excepción se usa la palabra reservada "throw", seguida de la creación de una instancia de la excepción a lanzar.

```
public static void main(String[] args) throws IngresoIncorrecto {
    int edad;
    Scanner sc = new Scanner( source: System.in);
    System.out.print( s: "Ingrese su edad: ");
    edad = sc.nextInt();
    if (edad < 1 || edad > 120)
        throw new IngresoIncorrecto( message: "Edad incorrecta");
    else
        System.out.println( x: "Bienvenido");
}
```

```
Ingrese su edad: 0
Exception in thread "main" segundopaquete.IngresoIncorrecto: Edad incorrecta
        at segundopaquete.NewMain4.main(NewMain4.java:16)
```

Cláusula "throws" que habilita el uso de la excepción en el método.

Lanzamiento de la excepción.

Propagación de las Excepciones

idat

- ✓ La excepción es capturada por los métodos que invocaron al método que la lanzó.

```
public static void main(String[] args) throws IngresoIncorrecto{
    int edad;
    Scanner sc = new Scanner( source: System.in);
    System.out.print( s: "Ingrese su edad: ");
    edad = sc.nextInt();
    if (validar(edad))
        System.out.println( x: "Bienvenido");
}
static boolean validar(int edad) throws IngresoIncorrecto {
    if (edad < 1 || edad > 120)
        throw new IngresoIncorrecto( message: "Edad incorrecta");
    else
        return true;
}
```

```
Ingrese su edad: 0
Exception in thread "main" segundopaquete.IngresoIncorrecto: Edad incorrecta
        at segundopaquete.NewMain4.validar(NewMain4.java:20)
        at segundopaquete.NewMain4.main(NewMain4.java:15)
```

Cláusula “throws” que habilita el uso de la excepción en el método.

Excepción propagada al método “main”, pero no gestionada.

Captura y Gestión de las Excepciones

idat

```
public static void main(String[] args) throws IngresoIncorrecto{
    int edad;
    Scanner sc = new Scanner( source: System.in);
    System.out.print( s: "Ingrese su edad: ");
    edad = sc.nextInt();
    try {
        if (validar(edad))
            System.out.println( x: "Bienvenido");
    } catch (IngresoIncorrecto ii) {
        System.out.println( x: ii.getMessage());
    }
}
static boolean validar(int edad) throws IngresoIncorrecto {
    if (edad < 1 || edad > 120)
        throw new IngresoIncorrecto( message: "Edad incorrecta");
    else
        return true;
}
```

```
Ingrese su edad: 0
Edad incorrecta
```

Gestión de la Excepción.

Excepción propagada al método “main”-

Aplicación

Revisar ejemplos y realizar ejercicios prácticos

idat

Término

Indicaciones generales y/o Resumen de Sesión

idat

Resumen de Sesión

idat

- Arreglos
- Colecciones
- Gestión de Excepciones
- Sentencia try ... catch ... finally
- Creación de Excepciones
- Ejemplos y Ejercicios

GRACIAS