Centro de Investigación en Computación del IPN

Java - curso básico

Dra. Erandi Castillo Montiel

Array de objetos

- Debido al polimorfismo es posible que un array contenga referencias a objetos de distintas clases.
  - Superclase y todas sus subclases.

Ejemplo:

```
        Figura[] figuras = new Figura[3];
        figuras[0]= new Rectangulo();
        figuras[1]= new Circulo();
        figuras[2] = new Poligono();

for (int i =0 ; i<figuras.length ; i++){
        figuras[i].borrar();
}
```

Interfaces

- Va más allá de una clase abstracta, es una clase donde todos los métodos son abstractos, permite al desarrollador establecer el comportamiento que tendrán las clases que implementen la interfaz.

Sintaxis:

```
interface NombreInterfaz {
    tiporetorno nombreMetodo(argumentos);
}
```

Interface vs clase abstracta

- Interfaz es una lista de métodos no implementados, puede incluir declaración de constantes.
- Clase abstracta puede incluir métodos implementados y no implementados o abstractos, así como atributos.

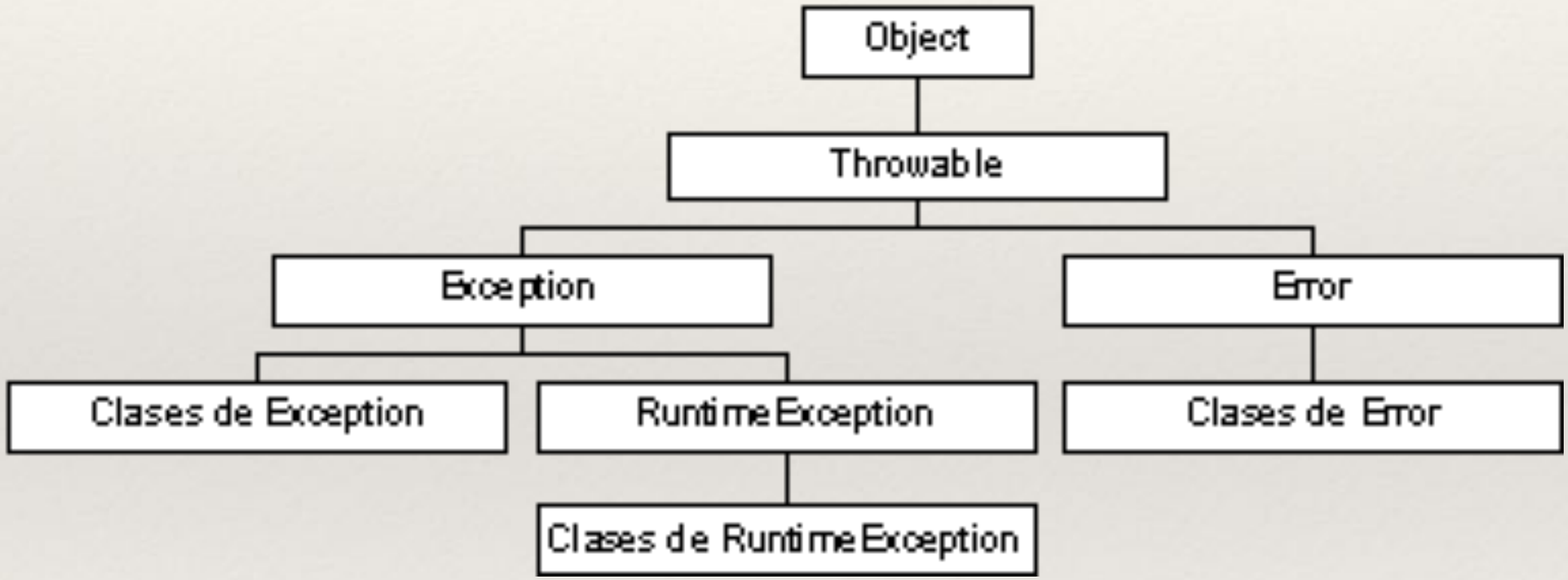
Tipo de excepciones

- *Error*: Excepciones que indican problemas muy graves, que suelen ser no recuperables y no deben casi nunca ser capturadas.

- *Exception*: Excepciones no definitivas, pero que se detectan fuera del tiempo de ejecución.

- *RuntimeException*: Excepciones que se dan durante la ejecución del programa.

Manejo de excepciones



- Para que el sistema de gestión de excepciones funcione, se ha de trabajar en dos partes de los programas:

  – Definir qué partes de los programas crean una excepción y bajo qué condiciones. Para ello se utilizan las palabras reservadas *throw* y *throws*.

  – Comprobar en ciertas partes de los programas si una excepción se ha producido, y actuar en consecuencia. Para ello se utilizan las palabras reservadas *try*, *catch* y *finally*.

Sintaxis

```
try{
        // sentencias que posiblemente generen excepciones.
}catch( tipo_de_excepción ){
        // código para solucionar la excepción.
}
```

Ejemplo

```java
public class Cbasico {
    public static void main(String[] args) {
        // TODO code application logic here

        System.out.println(Cbasico.dividir(30,0));
        System.out.println("Terminó bien el programa...");

    }

    public static double dividir(int dividendo, int divisor){
        return dividendo/divisor;
    }
}
```

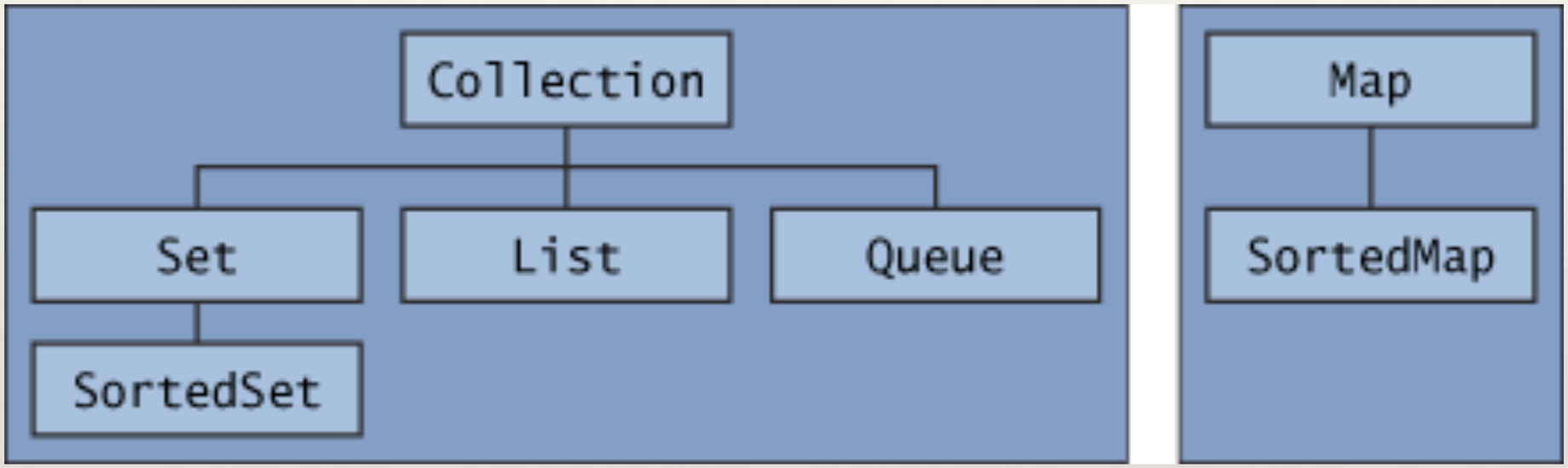
```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at cbasico.Cbasico.dividir(Cbasico.java:14)
        at cbasico.Cbasico.main(Cbasico.java:7)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
public double dividir(int dividendo, int divisor){
    try{
    return dividendo/divisor;
    }catch (ArithmeticException ex){
        System.out.println("ArithmeticException" + ex.getMessage());
    }
    return 0;
}
```

Colecciones

- Una colección es un objeto que almacena un conjunto de referencias a otros objetos, las colecciones son dinámicas, permiten agregar o quitar objetos en tiempo de ejecución.
- Operaciones básicas:
  - Agregar objetos
  - Eliminar objetos
  - Obtener un objeto
  - Localizar un objeto
  - Iterar a través de la colección

Interfaces



Interfaces

- *Collection:* representa un grupo de objetos, se usa cuando se desea manipular la colección con el máximo de generalidad deseado.
- *Set:* Una colección que no puede tener elementos duplicados.
- *List:* Una colección ordenada. Las Listas pueden contener elementos duplicados. Es posible acceder a los elementos por su índice.
- *Queue:* Una colección utilizada para guardar varios elementos por prioridad, cada elemento de la cola debe especificar sus propiedades de ordenamiento.
- *Map:* un objeto que trabaja por parejas siendo sus componentes, clave y valor, no permite tener claves duplicadas, cada clave debe corresponder al menos un valor.
- *SortedSet:* Mantiene sus elementos en orden ascendente.
- *SortedMap:* Mantiene sus mapeos en orden ascendente por clave

Conjuntos -Set

- Set. Interface encargada del tratamiento los objetos como conjuntos.
- SortedSet. Interface que permite que los elementos del conjunto estén ordenados completamente.

HashSet

- Clase que implementa la interfase Set
- Utiliza tablas hash (campo “clave”) para la localización de objetos
- La posición del objeto esta determinada por el campo clave

```
// HashSet
Set s1 = new HashSet();
s1.add("Zara");
s1.add("Mahnaz");
s1.add("Ayan");
s1.add(null);
System.out.println();
System.out.println(" Set Elements");
System.out.print("\t" + s1);
```

TreeSet

- Es la clase que implementa la interface SortedSet basada en el uso de la estructura de un árbol.
- Permite que los elementos estén ordenados por orden natural o por orden definido por un comparados.

```
        // Create a tree set
TreeSet ts = new TreeSet();

// Add elements to the tree set
ts.add("C");
ts.add("A");
ts.add("B");
ts.add("E");
ts.add("F");
ts.add("D");
System.out.println(ts);
```

Listas características

- Permiten acceso aleatorio
- Están ordenadas
- Fácilmente se pueden agregar/eliminar elementos sin ninguna restricción.
- Sigue los patrones de un Array.

Tipos de listas

- ArrayList: buen performance en el acceso a elementos, parecido a un arreglo.
- LinkedList: Más rápida que un arraylist añadiendo elementos al inicio y eliminando elementos en general.

ArrayList

- Representa una colección basada en índices, donde cero es la posición del primer elemento.
- Sintaxis:

```
ArrayList  nombre_array = new ArrayList();
```

Algunos métodos del ArrayList

boolean add(Object ob) : añade un objeto a la colección y lo sitúa al final de la misma. Devuelve true si logra agregarlo.

```
    // ArrayList
ArrayList a1 = new ArrayList();
a1.add(1);
a1.add(2);
a1.add(3);
for(Object x:a1){
    System.out.println((Integer)x);
}
```

- void add(int índice, Object o): añade el objeto al arrayList en la posición índice, desplazando hacia adelante el resto de los elementos.
- Object get(int índice): devuelve el objeto que ocupa la posición índice, Tomar en cuenta que el método devuelve una referencia tipo Object, para tener su correcta referencia es necesario hacer una conversión explicita.
- Object remove(int índice): remueve el objeto que ocupa la posición índice, regresando el objeto.
- void clear(): elimina todos los objetos de la colección.
- int indexOf(Object o): regresa el índice que ocupa el objeto del argumento, -1 si el objeto no se encuentra.
- Int size(): regresa el tamaño del arrayList.

```java
public static void main(String[] args) {

    LinkedList ll = new LinkedList();

    ll.add("F");
    ll.add("B");
    ll.add("D");
    ll.add("E");
    ll.add("C");
    ll.addLast("Z");
    ll.addFirst("A");
    ll.add(1, "A2");
    System.out.println("Contenido original de ll: " + ll);

    ll.remove("F");
    ll.remove(2);
    System.out.println("Contenido despues de borrar : "
     + ll);

    ll.removeFirst();
    ll.removeLast();
    System.out.println("ll despues de borrar primero y ultimo: "
     + ll);

    Object val = ll.get(2);
    ll.set(2, (String) val + " Cambiado");
    System.out.println("ll despues de cambio: " + ll);

}
```

Map: características

- Dada una tupla (llave, valor), es posible almacenar la tupla en un objeto Map, después obtener el valor, usando la llave.

- La utilización de colecciones basadas en llaves, resulta útil, en aplicaciones en las que se requiera realizar búsquedas de objetos a partir de un dato que lo identifica.

Tipos de Map

- HashMap: implementación genérica, no concurrente, no ordena llaves.
- LinkedHashMap: garantiza orden en las llaves por tiempo de inserción. Lento en inserción y eliminación.
- TreeMap: Ordena las llaves del map.
- ConcurrentHasMap: un map concurrente, no permite valores nulos.

HashMap

```
public static void main(String[] args) {
    Map m1 = new HashMap();
    m1.put("Tania", "8");
    m1.put("Ana", "31");
    m1.put("Diana", "12");
    m1.put("Juana", "14");
    System.out.println();
    System.out.println(" elementos del Map");
    System.out.print("\t" + m1);
}
```

```
Integer llave = 14;

String valor = (String) m1.get(llave);
System.out.println(" \n llave: " + llave +" valor: "+ valor);

Iterator llaveIterador = m1.keySet().iterator();
while(llaveIterador.hasNext()){
  Integer key =(Integer) llaveIterador.next();
    System.out.println("key: " + key + " valor: " + m1.get(key));
  }

System.out.println("HashMap contiene la llave 21 : " + m1.containsKey(31));
System.out.println("HashMap contiene el valor 21 : " + m1.containsValue(31));
System.out.println("HashMap contiene el valor Juana: " + m1.containsValue("Juana"));
}
```

Colecciones de tipos genéricos

- Permite al compilador saber que tipo de objetos van a ser almacenados en la colección.
- Genera un mensaje al momento de compilar el código fuente.
- No se hace necesario realizar una conversión explicita durante su recuperación.

Sintaxis:

```
tipo_colección <tipo_objeto> variable;
variable = new  tipo_colección<tipo_objeto>();
```

```java
ArrayList<String> lista;
lista = new ArrayList<String>();
lista.add("cadena uno");
lista.add("cadena dos");
lista.add("cadena tres");
```

```java
for(String cad : lista){
    System.out.println(cad);
}
```

Algoritmos: Clase Collection

- sort: Ordena una lista de elementos en orden ascendente

```
Collections.sort(lista);
System.out.println(lista);
```

- shuffle: Desordena la lista de forma aleatoria,

```
Collections.shuffle(lista);
System.out.println(lista);
```

- binarySearch: Busca un elemento en específico en una lista ordenada. Da por hecho que la lista se encuentra ordenada en orden ascendente.

```
int pos = Collections.binarySearch(lista, "cadena cinco");
System.out.println("posicion :" +pos);
```

```
Collections.sort(l1);
System.out.println("\t" + l1);
int pos=Collections.binarySearch(l1, "D");
System.out.println("posicion "+ pos);
Collections.shuffle(l1);
System.out.println("\t"+ l1);
```

Valores extremos

- min: devuelve el elemento mínimo de una colección de acuerdo a su orden natural.
- max: devuelve el máximo de los elementos.

```
System.out.println("maximo "+ Collections.max(l1));
System.out.println("minimo "+ Collections.min(l1));
```

Acceso a disco

- Los archivos del disco se representan de forma lógica en las aplicaciones Java como objetos de la clase File.
- La clase File (java.io) no se utiliza para transferir datos entre las aplicaciones y el disco, se usa para obtener información sobre los archivos, o la creación y eliminación de los mismos.

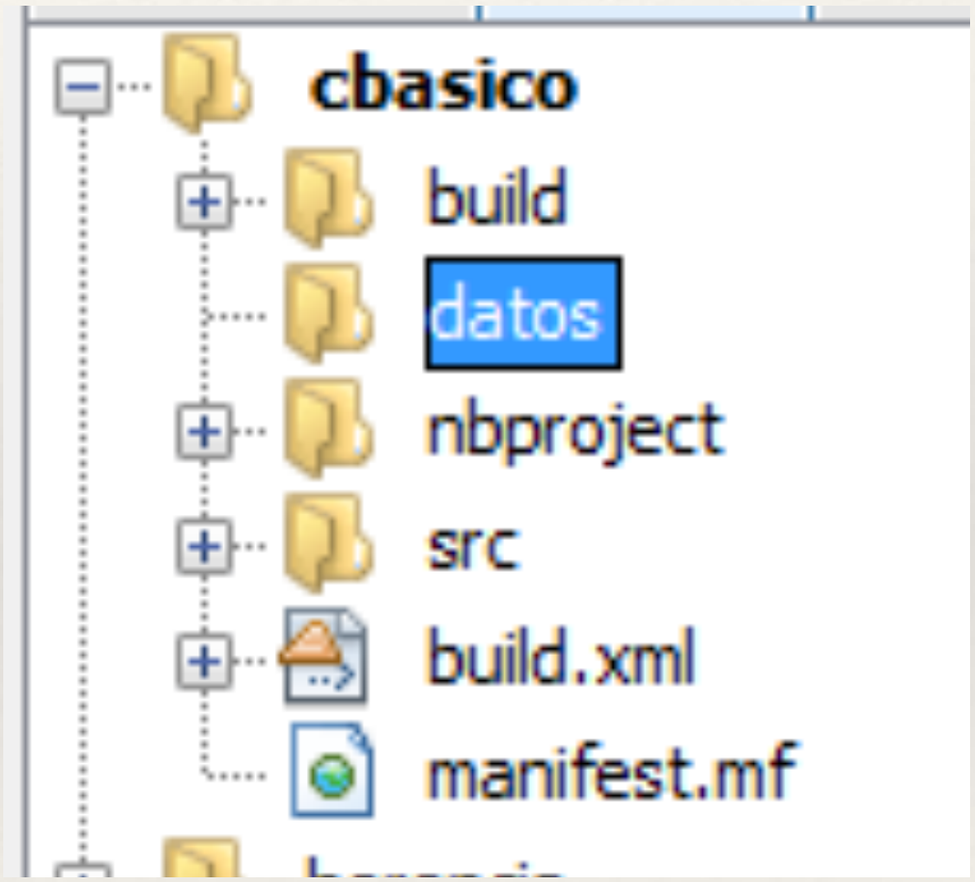
Sintaxis:

```
File archivo = new File("nombrearchivo.ext");
File arch2 = new File("directorios…");
File arch3 = new File(File dir, String nombre_archivo);
```

- Crear un objeto File, no implica que exista el archivo, o el directorio, si el archivo no existe, no se lanzan excepciones, tampoco se crea de forma implícita.
- boolean createNewFile();crea el archivo indicado en el constructor, devuelve true, si se ha podido crear el archivo, false si el archivo ya existía y no se ha creado.
- boolean mkdir(); Crea el directorio cuyo nombre ha sido especificado en el constructor, devuelve true si el directorio no existía y se ha podido crear, false en caso contrario.

```
File dir = new File("datos");
dir.mkdir();
File archivo = new File(dir,"info.txt");
```

Llamar a:
archivo.createNewFile();



Información sobre un archivo/directorio

- boolean canRead(); indica si es posible leer o no el archivo.
- boolean canWrite(); si es posible o no escribir.
- boolean exists(); Indica si existe o no el archivo o directorio indicado.
- boolean isFile(); Indica si el objeto File hace referencia a un archivo.
- boolean isDirectory();
- String getName();
- String getAbsolutePath();

```
System.out.println("puede leer "+archivo.canRead());
System.out.println("puede escribir "+archivo.canWrite());
System.out.println("existe "+archivo.exists());
System.out.println("es un archivo "+archivo.isFile());
System.out.println("es un directorio "+archivo.isDirectory());
System.out.println("su nombre "+archivo.getName());
System.out.println("su ruta absoluta "+archivo.getAbsolutePath());
```

Borrado y renombrado

- boolean delete(); elimina el archivo o directorio al que referencia el objeto, devuelve true, si lo logró eliminar, false en caso contrario.
- boolean renameTo(File nuevo); Renombra el archivo o directorio asignándole el nombre del objeto File especificado.

```
File dir = new File("datos");
dir.mkdir();
File archivo = new File(dir,"info.txt");
try{
    archivo.createNewFile();
}catch(IOException e){
    e.printStackTrace();
}
File f2= new File(dir,"datosNuevos.txt");
archivo.renameTo(f2);
```

Lectura de un archivo de texto

- Para lectura de un archivo de texto, Java proporciona dos clases, pertenecientes al paquete java.io
- FileReader y BufferReader.
- Para recuperar información de un archivo es necesario crear un objeto FileReader asociado al archivo.

```
archivo.renameTo(f2);
try{
    FileReader fr = new FileReader(archivo);
}catch(FileNotFoundException e){
    e.printStackTrace();
}
```

- FileReader proporciona el método read() para leer información del archivo, recuperando la información como tipo byte.
- Para facilitar el tratamiento de la información se recurre a la clase BufferReader.

```
FileReader fr;
BufferedReader br;
try{
    fr = new FileReader(archivo);
    br = new BufferedReader(fr);
}catch(FileNotFoundException e){
    e.printStackTrace();
}
```

```
File dir = new File("datos");
dir.mkdir();
File archivo = new File(dir,"info.txt");
try{
    archivo.createNewFile();
}catch(IOException e){
    e.printStackTrace();
}
FileReader fr;
BufferedReader br;
try{
    fr = new FileReader(archivo);
    br = new BufferedReader(fr);
    String cad=null;
    while((cad=br.readLine())!=null){
        System.out.println(cad);
    }
}catch(FileNotFoundException e){
    e.printStackTrace();
}catch(IOException e){
    e.printStackTrace();
}
```

Escritura en un archivo de texto

- Al igual que el caso de la lectura, la escritura a archivos se lleva a cabo en dos pasos.
- Se utilizan las clases, FileWriter y PrintWriter

```
try{
    FileWriter fw = new FileWriter(archivo,true);
    PrintWriter out = new PrintWriter(fw);
}catch(IOException e){
    e.printStackTrace();
}
```

```
String[] cadenas = {"Nueva cadena","Otra cadena para guardar",
                    "una cadema más" , "Mas cadenas para guardar"};
try{
    FileWriter fw = new FileWriter(archivo,true);
    PrintWriter out = new PrintWriter(fw);

    for(String cad : cadenas){
        out.println(cad);
    }
    out.flush();
    out.close();
}catch(IOException e){
    e.printStackTrace();
}
```

Escritura de objetos en archivos

- Para que un objeto pueda ser almacenado en un archivo, es necesario que la clase a la que pertenece sea "serializable". Que se obtiene implementando la interfaz, java.io.Serializable.
- La escritura a disco se realiza con las clases, FileOutputStream y ObjectOutputStream.

```java
 */
public class Persona implements Serializable{
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad){
        this.nombre=nombre;
        this.edad = edad;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

```
File arch3 = new File(dir,"datosObjetos.obj");
try{
    arch3.createNewFile();
    FileOutputStream fos = new FileOutputStream(arch3,false);
    ObjectOutputStream os = new ObjectOutputStream(fos);
    os.writeObject(new Persona("Iliac",25));
    os.writeObject(new Persona("Ariadna",2));

    os.flush();
    os.close();
}catch(IOException e){
    e.printStackTrace();
}
```

Lectura de libros

```
try{
    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(arch3));
    while(true){
        Persona p = (Persona)ois.readObject();
        System.out.println(p.getNombre()+":"+p.getEdad());
    }
}catch(FileNotFoundException e){
    e.printStackTrace();
}catch (EOFException ex) {
        System.out.println("Se alcanzó el final del archivo");
} catch (IOException ex) {
        ex.printStackTrace();
} catch (ClassNotFoundException ex) {
        ex.printStackTrace();
}
```