**1)Explicacion de Documentación**

En la carpeta del dia de hoy cuyo nombre es **clase12\_14mayo** se encuentran los ejercicios realizados el dia de hoy como nuestra primera clase de java.

El IDE utilizado es Netbeans, para poder utilizarlo se debe tener el notebook instalado jave.

JDK

**2)Reflexion**

La clase fue muy didáctica, a un nivel cómodo para nuestro aprendizaje, en particular no tuve problemas para realizar los ejercicios.

Tengo un poco de dudas respecto a un constructor que creo el profesor ,pero explico que lo explicará en detalle mas adelante.

**3)Aprendizaje extra.**

En particular me puse a revisar lo que son las excepciones en java.

Se adjunta información que leí luego de la clase

***Excepciones en java***

Las excepciones permiten que un método informe al código que lo ha invocado acerca de algún error o situación anómala que se haya producido durante su ejecución.

Por ejemplo, supongamos un método que recibe un código postal y devuelve el nombre del ayuntamiento al cual pertenece. En el caso normal de que el código postal sea correcto, el método devolvería el nombre del ayuntamiento. En el caso de que el código postal que se le haya pasado no exista o su formato sea incorrecto, el método lo notificaría *lanzando* una excepción. Una excepción en Java es un objeto de la [clase Exception](http://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html) o de alguna de sus subclases.

## **Declaración de nuevas excepciones**

Java cuenta con muchas excepciones predefinidas para determinadas situaciones, como por ejemplo [IOException](http://docs.oracle.com/javase/7/docs/api/java/io/IOException.html) para errores producidos en operaciones de entrada/salida, como es el caso de la lectura de ficheros.

Para nuestros propios programas, a veces es útil que creemos nuestra excepción a medida. Para ello, tenemos que declarar una nueva clase que herede de Exception. Por ejemplo:

public class BadPostCodeException extends Exception {

public BadPostCodeException() {

super();

}

public BadPostCodeException(String message) {

super(message);

}

}

## **Cómo se lanza una excepción**

Cuando en un método necesitamos lanzar una excepción, utilizaremos la palabra clave throw, seguida de una instancia de la excepción a lanzar. Por ejemplo:

if (postCode.length() != 5) {

throw new BadPostCodeException("Postcodes must have 5 digits");

}

for (int i = 0; i < postCode.length(); i++) {

if (!Character.isDigit(postCode.charAt(i))) {

throw new BadPostCodeException("Postcodes can only contain digits");

}

}

City city = lookupPostCode(postCode);

if (city == null) {

throw new BadPostCodeException("Postcode not in database: " + postCode);

}

Salvo en el caso de un grupo especial de excepciones, en general un método que pueda potencialmente lanzar una excepción debe indicarlo explícitamente mediante la palabra clave throws (nótese la "s" final, no confundir con throw) seguida del nombre de la clase de la excepción que puede lanzar:

public class PostCodeManager {

(...)

public String town(String postCode) throws BadPostCodeException {

if (postCode.length() != 5) {

throw new BadPostCodeException("Postcodes must have 5 digits");

}

for (int i = 0; i < postCode.length(); i++) {

if (!Character.isDigit(postCode.charAt(i))) {

throw new BadPostCodeException("Postcodes can only contain digits");

}

}

City city = lookupPostCode(postCode);

if (city == null) {

throw new BadPostCodeException("Postcode not in database: " + postCode);

}

(...)

}

(...)

}

Si el método puede lanzar más de una clase de excepción, se ponen los nombres de todas las clases separados por comas tras un único throws.

## **Qué ocurre cuando se lanza una excepción**

Cuando se produce una excepción, la máquina virtual interrumpe la ejecución normal del programa y busca un bloque de código adecuado para tratar la situación. Si no encuentra este código en el método actual, la excepción se propaga hacia el método que lo haya invocado y se busca allí el código que la trate. Si tampoco ese método dispone del código adecuado, se propagará a su vez al que lo haya invocado, y así sucesivamente.

En este apartado se explica el caso en que un método no dispone de código adecuado para tratar una excepción. Como se ha explicado, en este caso la excepción se propaga al método que lo haya invocado. Un método que no proporcione código para tratar la excepción debe declarar que puede lanzar la excepción con la palabra clave throws al igual que se explica en el apartado anterior:

public class Location {

(...)

private String postCode;

public String townAsString() throws BadPostCodeException {

PostCodeManager manager = new PostCodeManager();

String town = manager.town(postCode);

return postCode + " " + town;

}

(...)

}

En el ejemplo, la invocación al método town puede potencialmente lanzar una excepción. El método townAsString no está proporcionando código adecuado para tratarla. Por tanto, la excepción se propagaría a través de él, y entonces también debe declarar que puede lanzarla, como se ve en el ejemplo.

Si en la llamada a town surge una excepción, esta se propaga al método que haya invocado a townAsString.

La excepción continuará propagándose por la pila de invocaciones hasta que en algún método sea tratada. Si se alcanza el método main, y este tampoco proporciona ningún código para tratarla, la máquina virtual cortará la ejecución del programa y mostrará al usuario, normalmente en pantalla, el mensaje de la excepción y la ubicación del programa en que se haya producido.

## **Cómo se trata una excepción**

Un método puede decidir capturar una excepción si tiene sentido colocar en ese método el código que la trata adecuadamente. Para capturar una excepción se utiliza un bloque try/catch:

try {

// código en que podría surgir la excepción

} catch (ClaseDeLaExcepción e) {

// código que trata la situación de la excepción

}

En el momento en que surja la excepción en el bloque try, se corta su ejecución y se pasa a ejecutar el código del bloque catch. Una vez finaliza el bloque catch, se continúa con la ejecución del resto del código que siga a try/catch. También se continúa normalmente en ese punto si el bloque try finaliza sin que surjan excepciones.

A continuación se muestra una implementación alternativa de townAsString en que se captura la excepción:

public class Location {

(...)

private String postCode;

public String townAsString() {

String result;

PostCodeManager manager = new PostCodeManager();

try {

String town = manager.town(postCode);

result = postCode + " " + town;

} catch (BadPostCodeException e) {

result = "Unknown location at postcode " + postCode;

}

return result;

}

(...)

}

Si en el método town surge una excepción, se corta la ejecución del bloque try, y por tanto ya no se ejecuta la instrucción result = postCode + " " + town, sino que el programa salta al interior del bloque catch. La instrucción return result se ejecuta siempre, surja o no alguna excepción, porque está a continuación del bloque try/catch.

Nótese que el método townAsString no declara en este caso que se lance la excepción. El motivo es que al capturar la excepción, esta ya no puede propagarse a través del método.