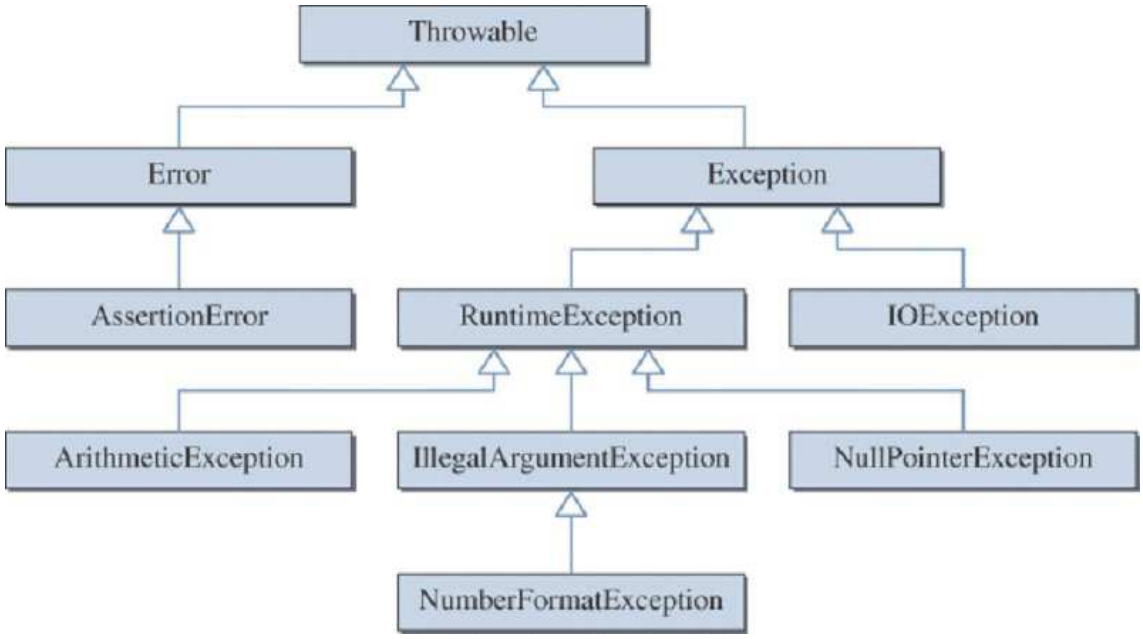
EXCEPCIONS

Una excepció és una situació anòmala que pot produir-se durant la execució d'un programa, com pot ser fer una divisió per zero, accedir a posicions fora dels límits d'un array o una lectura errònia d'entrada sortida.

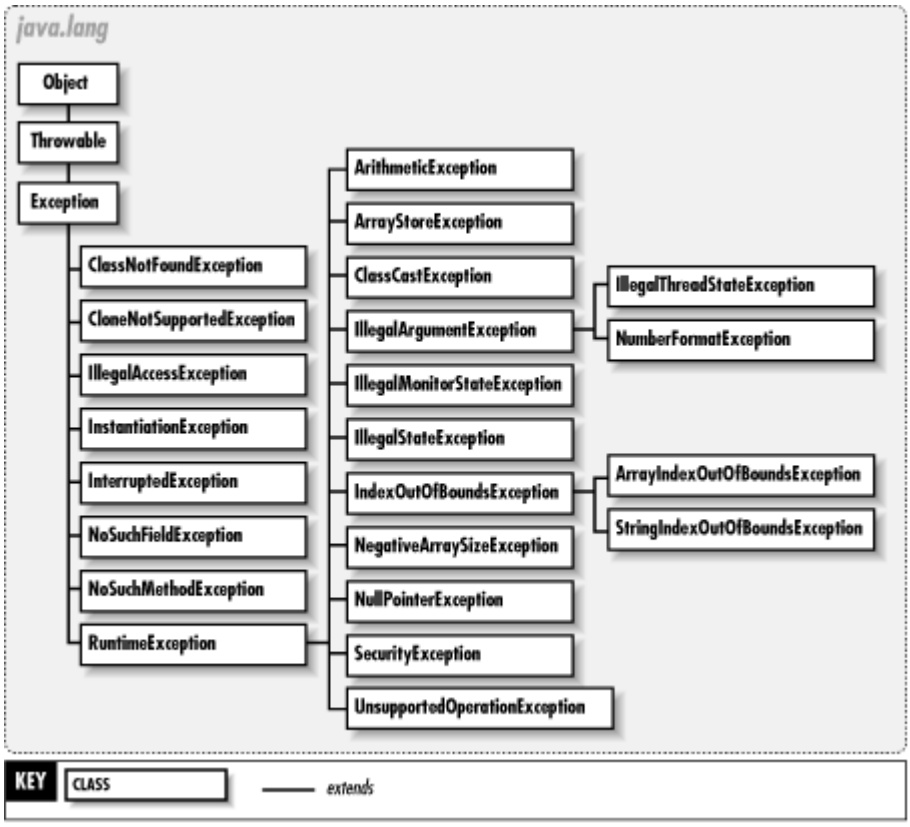
Java ens permet capturar aquests fets mitjançant les excepcions, permeten al programador definir el comportament del programa en cas que es produeixin aquestes situacions.

També es poden produir errors. Un error és una situació anormal irreversible com pot ser una fallida d'una màquina virtual java. Per regla general un programa no pot definir un comportament per recuperar-se ja que s'escapa del control del programa.

Cada tipus d'excepció està representada per una subclasse d'exception. La figura representa algunes de les excepcions existents. Els errors son subclasses de Error. Tant Exception com Error son subclasses de Throwable, aquesta última es subclasse de la classe Object.



Al produir-se una excepció en un programa, es crea un objecte de la classe Exception que podem fer servir per obtenir informació d'aquesta mateixa excepció. En la figura següent podem veure algunes de les més habituals.



Les excepcions es divideixen en dos tipus :

- **Marcades:** Son de captura obligatòria. Totes les classes excepte RuntimeException i les seves subclasses pertanyen a aquest tipus. Per exemple si tenim una excepció IOException aquesta pot ser llençada pel mètode readLine() de la classe BufferedReader.
    - Si en un bloc de codi s'invoca a un mètode que pot provocar una excepció marcada i aquesta no es captura, el programa no compilarà.
    - Per declarar una excepció s'utilitza la paraula throws, seguida de les excepcions que pot provocar:
        - Public String readLine() throws IOException
        - Public void service (...) trhows ServletException,IOException.

- **No marcades:** Son totes les excepcions que es produeixen en temps d'execució, les RuntimeExceptions i subclasses. No es obligatori capturar-les. La gran majoria de aquestes son degudes a una mala programació. Per aquest motiu la solució no es passa per capturar-les si no evitar que es produeixin. Només es recomana capturar les del tipus ArithmeticException. Per tant per tal de solucionar aquest problemes haurem de mirar la consola.

```
Exception in thread "main"

java.lang.ArithmeticException: / by zero
	at figura.AvioCombat.arrancar(AvioCombat.java:31)
	at figura.AvioCombat.main(AvioCombat.java:44)
```

Captura d'excepcions

El mecanisme de captura d'excepcions és el bloc try .. catch.. finally. És el bloc que permet capturar l'objecte excepció produït per una instrucció.

```
try {
// bloc d'accions on es pot produir la excepció
} catch (TipusExcepcio1 arg) {
   // tractament excepció 1
} catch (TipusExcepcio2 arg) {
   // tractament excepció 2
} finally {
 // Instruccions d'última execució
}
```

- **Try:** Delimita les instruccions on es pot produir l'excepció, si es produeix, el control del programa salta al bloc catch definit per el tipus de excepció que s'ha produït, passant com a paràmetre la excepció llençada. Opcionalment es pot incloure un bloc finally on es pot definir un bloc d'instruccions de obligatòria execució.
- **Catch:** Defineix les instruccions que es deurien executar en cas de que es produeixi una determinada excepció. S'han de tenir en conte les següents regles:
    - Es poden definir tants blocs catch com es consideri necessari. Cada bloc servirà per tractar un determinat tipus d'excepció. No pot haver dos o més catch amb la mateixa excepció declarada.
    - Un bloc catch s'utilitza per capturar qualsevol excepció que es correspongui al tipus declarat o qualsevol de les seves subclasses.
    - La cerca del bloc catch es fa de forma seqüencial, de forma que el primer catch coincident serà el que s'executarà. Un cop fet el bloc el control del programa es transfereix al bloc finally, i si no existeix a la instrucció següent al últim bloc catch, independentment de si hi ha un o més catch coincidents.

```
package excepcions;

public class exemple1 {

	public static void main(String [] args){

		try {
			int s=4/0;
			System.out.println("El programa continua");
		} catch (ArithmeticException e) {
			System.out.println("Divisió per zero");
		} catch (Exception e) {
			System.out.println("Excepció general");

		}
		   System.out.println("Final del main");
		}
	}
```

- Mai es torna el control al punt on es produeix la excepció.
- Els catch relacionats per herència més específics han d'estar situats per davant dels més genèrics. Si no es així es produeix un error de compilació.

Compila correctament	No compila correctament

`try {`
`…`
`} catch (ArithmeticException e) {`
`…`
`} catch (Exception e) {`
`…`
`}`
	`try {`
	`…`
	`} catch (Exception e) {`
	`…`
	`} catch (ArithmeticException e) {`
	`…`
	`}`

- Si es produeix una excepció no marcada per la que no s'ha definit bloc catch, aquesta serà propagada per la pila de crides fins trobar algun punt en el que es tracti l'excepció, de no existir tractament per la mateixa, la maquina virtual avortarà l'execució i enviarà el bolcat de la pila a la consola.
- Si hi ha un finally el catch es opcional. Si no hi ha finally es obligatori almenys un bloc catch.

- **Finally:** El seu us es opcional, s'executarà tant si s'ha produït una excepció com si no.

Propagació d'una excepció

En el cas de les excepcions no marcades es possible propagar-la sense capturar-la ja que no es obligatori. Per propagar-la sense capturar-la es suficient amb declarar-la en la capçalera del mètode on es pugui produir.

```
package excepcions;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class exemple2 {

	static void Imprimeix(BufferedReader bf) throws IOException{

		String n=bf.readLine(); //pot provocar una excepció

	}
	public static void main(String [] args){

		BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
		try {
			Imprimeix(b);
```

```
                } catch (IOException e) {
                        System.out.println("Error de lectura");
                }
        }
}
```

També podem llençar una excepció des de codi utilitzant l'expressió throw objecte_exception:

```
package excepcions;

public class Comte {
         private float saldo;
         public Comte(){
           saldo=0;
         }
         public void ingresar(float c){
           saldo+=c;
         }
         public void extreure(float c) throws Exception{
           if(saldo<c){
             //si no hi ha saldo suficient llencem la excepció
             throw new Exception();
           }
           else{
             saldo -= c;
           }
         }
         public float getSaldo(){
           return saldo;
         }
}
```

Mètodes pel control de una excepció

Totes les classes excepció hereten una sèrie de mètodes de Throwable que poden ser utilitzats a l'interior d'un catch per completar les accions del tractament de la excepció. Els mètodes més importants son:

- String getMessage(): Retorna un missatge de text associat a la excepció.
- Void printStackTrace(): Envia a la consola el bolcat de la pila associat a la excepció. Es força útil a la fase de desenvolupament de la aplicació per detectar errors a la programació.
- Void printStackTrace(PrintStream s): Permet enviar el bolcat a un objecte PrintStream, per exemple a un arxiu de log.

Classes d'excepció personalitzades

Quan un mètode necessita llençar una excepció com a forma de notificar una situació anòmala pot succeir que les classes ja existents no s'ajustin a les característiques que vulguem notificar. Per exemple en el cas de la classe Comte anterior no tindria molt de sentit llençar una excepció del tipus NullPointer o IOException quan es produeix una situació de saldo insuficient.

En aquests casos resulta més pràctic llençar una excepció personalitzada, subclasse Exception.

```
public class SaldoInsuficientException extends Exception {
  public SaldoInsuficientException()
  {
    //text predeterminat associat a la excepció
    super("Saldo negativo!!!!");
  }
  public SaldoInsuficientException(String s)
  {
    //permet personalitzar el text associat
    //a la excepció
    super(s);
  }
}

public void extraer(float c) throws SaldoInsuficienteException{
    if(saldo<c){
      //si no hi ha saldo suficient llença la excepció
      throw new SaldoInsuficienteException();
    }
    else{
      saldo -= c;
    }
  }
```

PRÀCTICA

Es pretén desenvolupar una aplicació que simuli el funcionament d'un caixer automàtic. Crea una classe que es digui comte que gestioni les operacions sobre el comte. A més dels constructors i camps que estimem necessaris, la classe contarà amb els següents mètodes:

- void ingressar(float c) per afegir diners.
- void extreure(float c) per treure diners.
- float getSaldo retorna el saldo actual.

Per altre costat existirà una classe amb un mètode main encarregada de la captura i presentació de dades i la gestió del compte. Al iniciar l'aplicació es mostrarà el següent menú.

1. Crear comte buit.
2. Crear comte amb saldo inicial.
3. Ingrés de diners
4. Treure diners
5. Veure saldo
6. Sortir

Per últim fes les modificacions necessàries per tal de afegir un mecanisme d'excepcions per tractar la situació de saldo insuficient. Si es vol treure més diners que el saldo disponible llençarem la excepció personalitzada SaldoInsuficientException. S'haurà de capturar al main.