# Programmazione 2

Quinta Esercitazione
12/04/2023

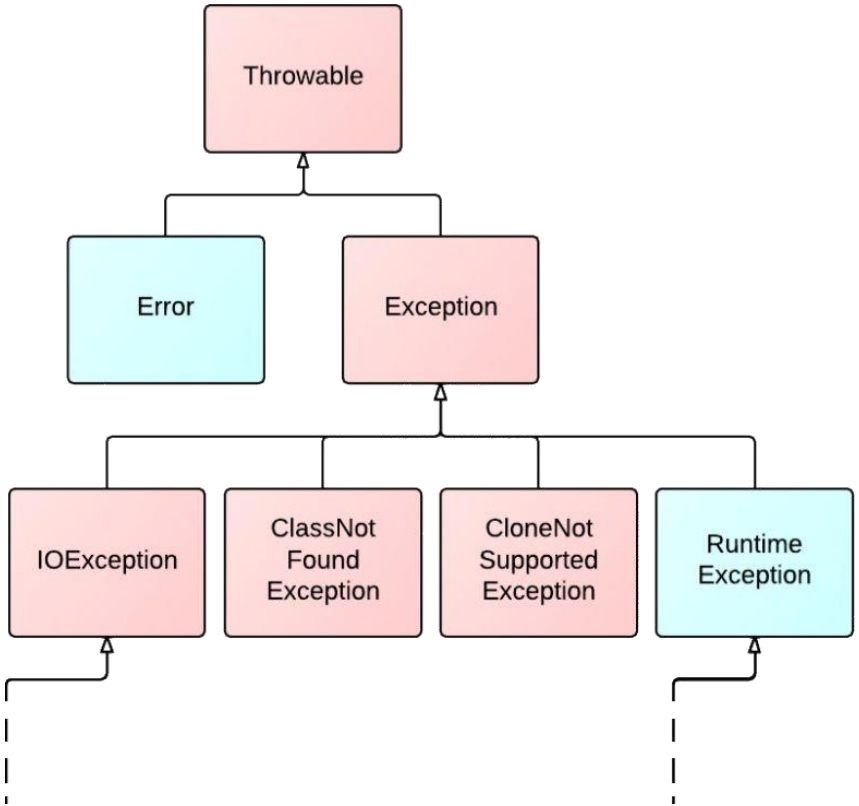
# Obiettivi dell’esercitazione

- Breve ripasso delle eccezioni e introduzione alle interfacce
- Esercizio di modellazione che riprenda questi concetti

# Ripasso: eccezioni

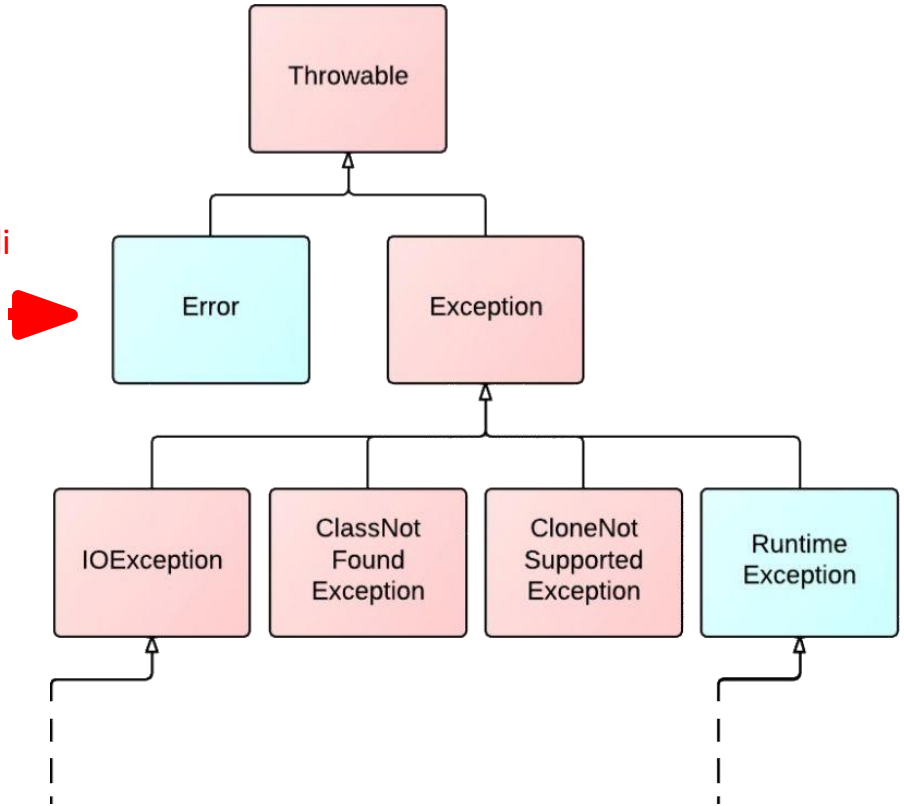
- Un'eccezione è un evento che occorre durante l'esecuzione di un programma
- Le eccezioni possono essere lanciate dai metodi (usando la keyword `throw`) e si propagano nei metodi chiamanti
- È possibile catturare un'eccezione (nel blocco `try - catch`) e fare qualcosa in risposta a questo evento (exception handling)

# Ripasso: eccezioni



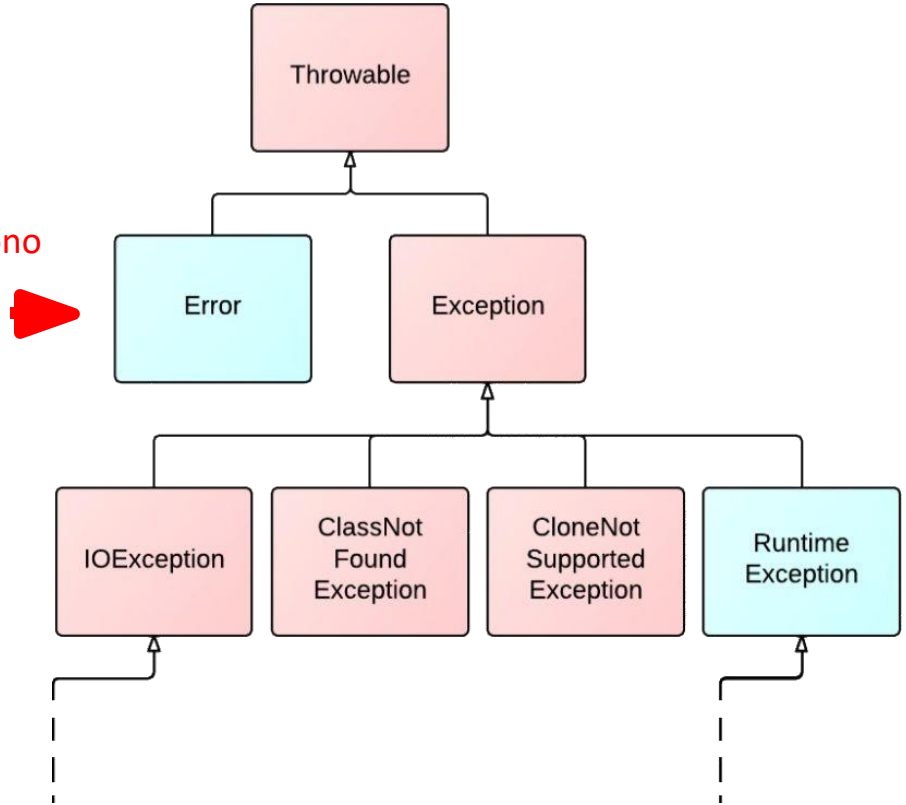
# Ripasso: eccezioni

Situazioni imprevedibili che pregiudicano l'esecuzione dell'interprete Java (es. è finita la RAM)

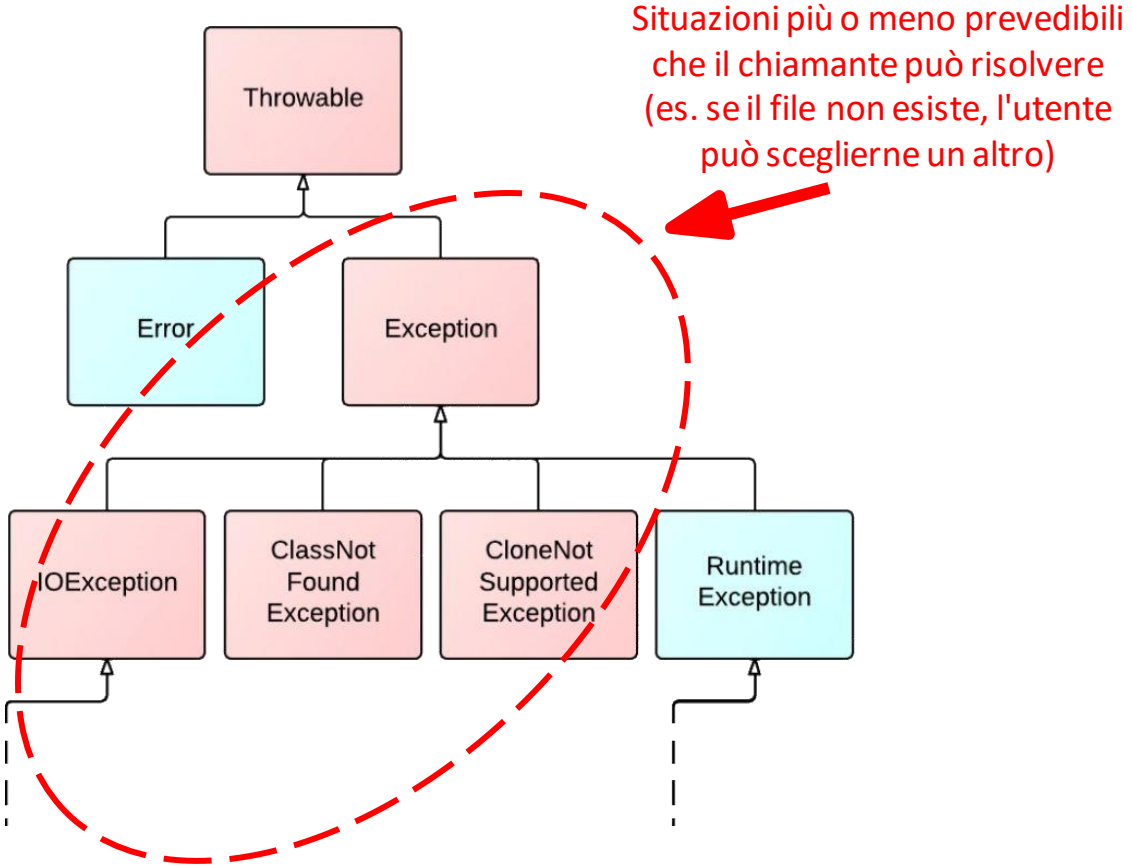


# Ripasso: eccezioni

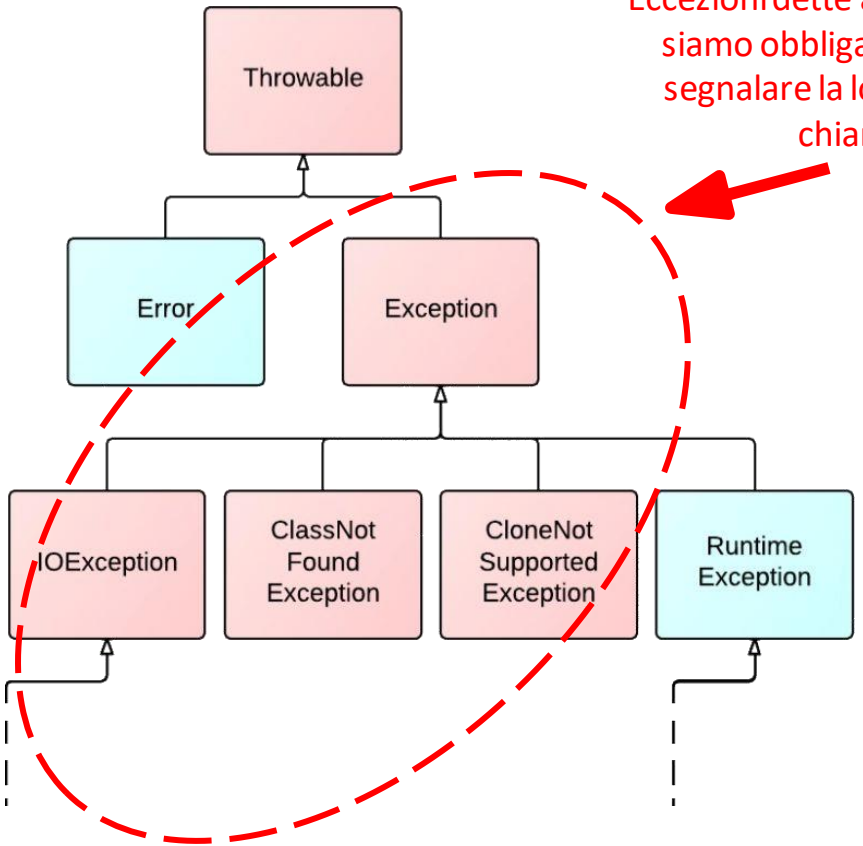
Errori per loro natura sono ingestibili



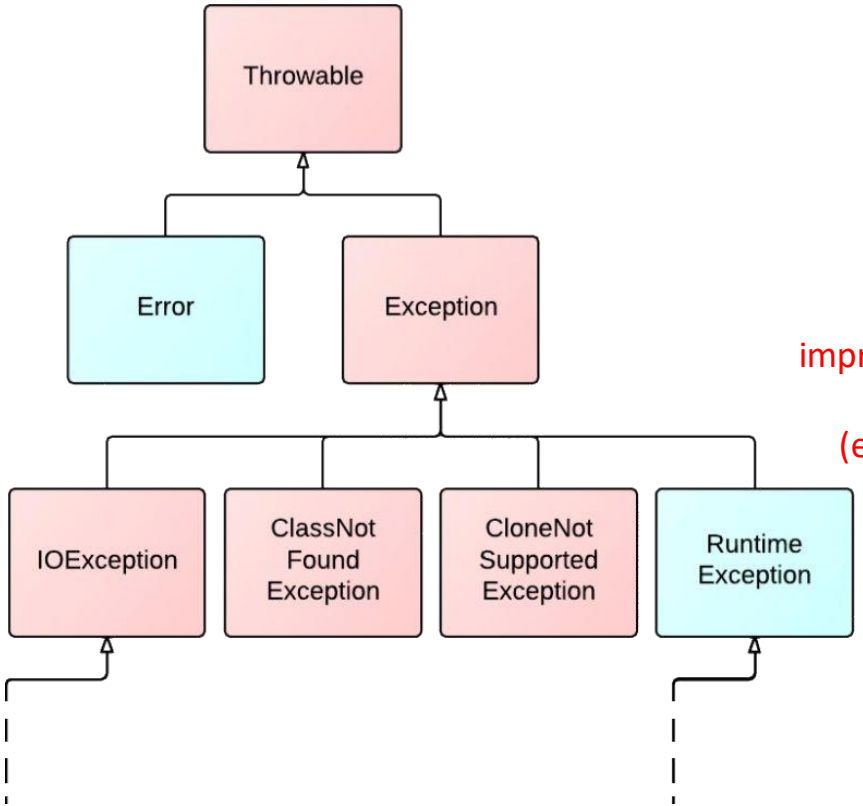
# Ripasso: eccezioni



# Ripasso: eccezioni

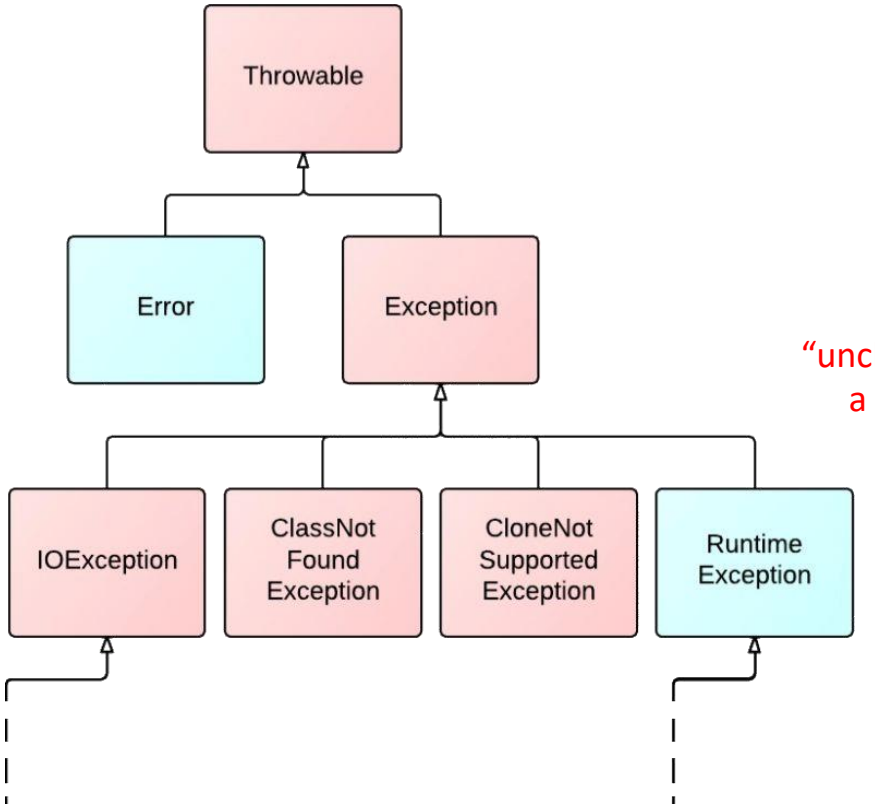


# Ripasso: eccezioni




Situazioni solitamente imprevedibili che indicano un bug nel codice del chiamante (es. il metodo ha ricevuto un oggetto null)

# Ripasso: eccezioni




Eccezioni dette anche "unchecked": non siamo obbligati a gestirle (ma possiamo, se vogliamo)

# Promemoria: come definire le eccezioni

- E’ possibile utilizzare le eccezioni già rese disponibili in Java, ma anche definirne di nuove

```
public class IndexOutOfBoundException extends RuntimeException {
    public IndexOutOfBoundException(String messaggioDiErrore){
        super(messaggioDiErrore);
    }
}
public class WriteException extends Exception {
    public IndexOutOfBoundException(){
        System.out.println( … );
    }
}
```

# Promemoria: come lanciare le eccezioni

```
public class IndexOutOfBoundException  extends  RuntimeException  { … }

public class WriteException  extends  Exception { … }

public class List {

     public int get(int i) {

     ... throw new IndexOutOfBoundException(); ...

     }

}

List l = new List();

int n = l.get(5);
```

compila, ma se si verifica la condizione dell'eccezione, l'esecuzione si blocca

# Promemoria: come gestire le eccezioni

```
public class IndexOutOfBoundException extends RuntimeException { … }
public class WriteException extends Exception { … }
public class List {
    public void writeToFile() throws WriteException {
        …  throw new WriteException();  …
    }
}
List l = new List();
l.writeToFile();
```

non compila

# Promemoria: come gestire le eccezioni

```
public class IndexOutOfBoundException extends RuntimeException { … }
public class WriteException extends Exception { … }
public class List {
    public void writeToFile() throws WriteException {
        …  throw new WriteException();  …
    }
}
List l = new List();
try {
    l.writeToFile();
} catch (WriteException e) { … }
```

compila, e l'esecuzione continua

# Promemoria: come gestire le eccezioni

```
public class IndexOutOfBoundException extends RuntimeException { … }
public class WriteException extends Exception { … }
public class List {
    public void writeToFile() throws WriteException {
        …  throw new WriteException();  …
    }
}
public static main void(String[] args) throws WriteException {
        List l = new List();
        l.writeToFile();
}
```

compila, e l'esecuzione continua

# Le interfacce

Nell'ingegneria del software ci sono situazioni in cui è necessario che diversi gruppi di programmatori si accordino su come i componenti software devono comunicare. Le **interfacce** definiscono questo accordo, permettendoci di separare l'implementazione dagli accordi.

In Java, le interfacce sono un **tipo di riferimento**, come le classi. Possono contenere:

- costanti (public, static e final di default)
- dichiarazioni di metodi (senza corpo, public di default)
- metodi statici

Le interfacce **NON possono essere istanziate**.

Le interfacce **possono essere estese** (da altre interfacce) o **essere implementate** (da altre classi)

Una classe **può implementare una o più interfacce**.

# Promemoria: come si definiscono le interfacce

```
public interface GroupedInterface extends Interface1, Interface2 {

    // dichiarazioni di costanti

    double E = 2.718282;

    // firme dei metodi

    void doSomething (int i, double x);

    int doSomethingElse(String s);

}

public class MyClass implements GroupedInterface{ ... }
```

Gli attributi sono automaticamente public static final, quindi queste parole chiave per convenzione non si scrivono

I metodi sono automaticamente public, per convenzione non si scrive

# Le interfacce: polimorfismo

Le interfacce consentono il polimorfismo, permettendo di trattare oggetti diversi con la stessa interfaccia:

```
public interface Veicolo{…}

public interface Auto extends Veicolo {…}

public class Automobile implements Auto{…}
```

Grazie al polimorfismo possiamo -> `Veicolo v = new Automobile();`

# Classi Astratte VS Interfacce

| Similarità | Diversità |
|---|---|
| Non possono essere istanziate | Le interfacce possono definire solo costanti pubbliche |
| Generalmente definiscono metodi senza implementazione | **Una classe può estendere una sola classe, ma implementare diverse interfacce** |
| | **Un'interfaccia può estendere tante interfacce** |

Usare le classi astratte se:

- volete condividere codice tra classi strettamente correlate
- le classi avranno uno stato comune, metodi privati comuni da condividere

Usare le interfacce se:

- le classi che la implementeranno non sono necessariamente correlate
- volete specificare solo il comportamento delle classi, senza preoccuparvi dell'implementazione

Fine quinta esercitazione