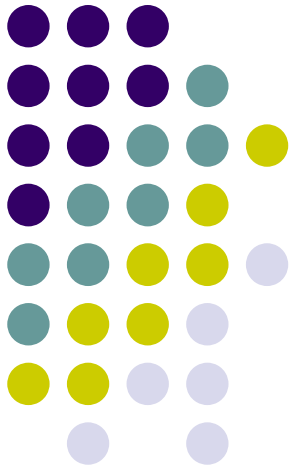
Università di Napoli Federico II - Dip. di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Corso di Laurea in Ingegneria Informatica

DIETI

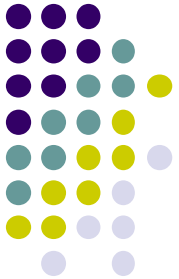
Corso di Programmazione

Gestione delle eccezioni

Concetti Avanzati



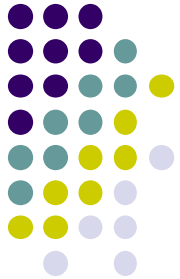
Introduzione alle Interruzioni



- Cos'è un'interruzione?
- Un'interruzione è un evento che interrompe il flusso normale di un programma.

- Eccezioni in Java
- Le eccezioni sono un modo per gestire errori o eventi inaspettati.
- Esempi di eccezioni: IOException, SQLException, NullPointerException.

Gestione delle Eccezioni con le Librerie Java



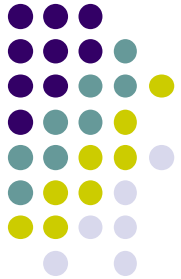
- Blocco try-catch
- Struttura di base per la gestione delle eccezioni.

```
try {
    // codice che può causare un'eccezione
} catch (TipoEccezione e) {
    // codice per gestire l'eccezione
}
```

- Blocco finally
- Utilizzato per eseguire codice che deve essere eseguito indipendentemente dal verificarsi dell'eccezione.

```
try {
    // codice che può causare un'eccezione
} catch (TipoEccezione e) {
    // codice per gestire l'eccezione
} finally {
    // codice che viene sempre eseguito
}
```

Creazione di Eccezioni Definite dall'Utente



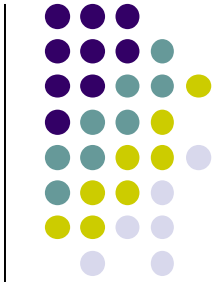
- Definire una nuova eccezione : le eccezioni personalizzate estendono la classe Exception o una delle sue sottoclassi.

```
public class MiaEccezione extends Exception {
    public MiaEccezione(String messaggio) {
        super(messaggio);
    }
}
```

- Utilizzare una eccezione definita dall'utente

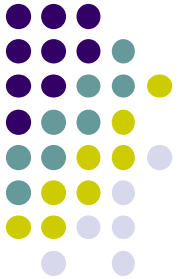
```
public class Esempio {
    public void metodo() throws MiaEccezione {
        // codice che può causare un'eccezione
        throw new MiaEccezione("Messaggio di errore");
    }
}
```

Esempi di Gestione delle Eccezioni



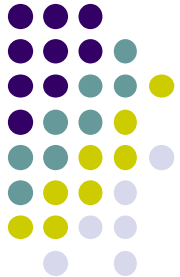
```
public class Esempio {
    // Metodo principale che viene eseguito all'avvio del programma
    public static void main(String[] args) {
        try {
            // Chiama il metodo 'metodo' che lancia un'eccezione di tipo 'MiaEccezione'
            metodo();
        } catch (MiaEccezione e) {
            // Cattura l'eccezione 'MiaEccezione' lanciata dal metodo
            // e stampa il messaggio dell'eccezione
            System.out.println("Eccezione catturata: " + e.getMessage());
        }
    }

    // Metodo statico che lancia un'eccezione di tipo 'MiaEccezione'
    public static void metodo() throws MiaEccezione {
        // Lancia un'eccezione di tipo 'MiaEccezione' con un messaggio personalizzato
        throw new MiaEccezione("Errore personalizzato");
    }
}

// Classe personalizzata per definire un'eccezione specifica
class MiaEccezione extends Exception {
    // Costruttore della classe 'MiaEccezione' che accetta un messaggio di errore
    public MiaEccezione(String message) {
        super(message);
    }
}
```

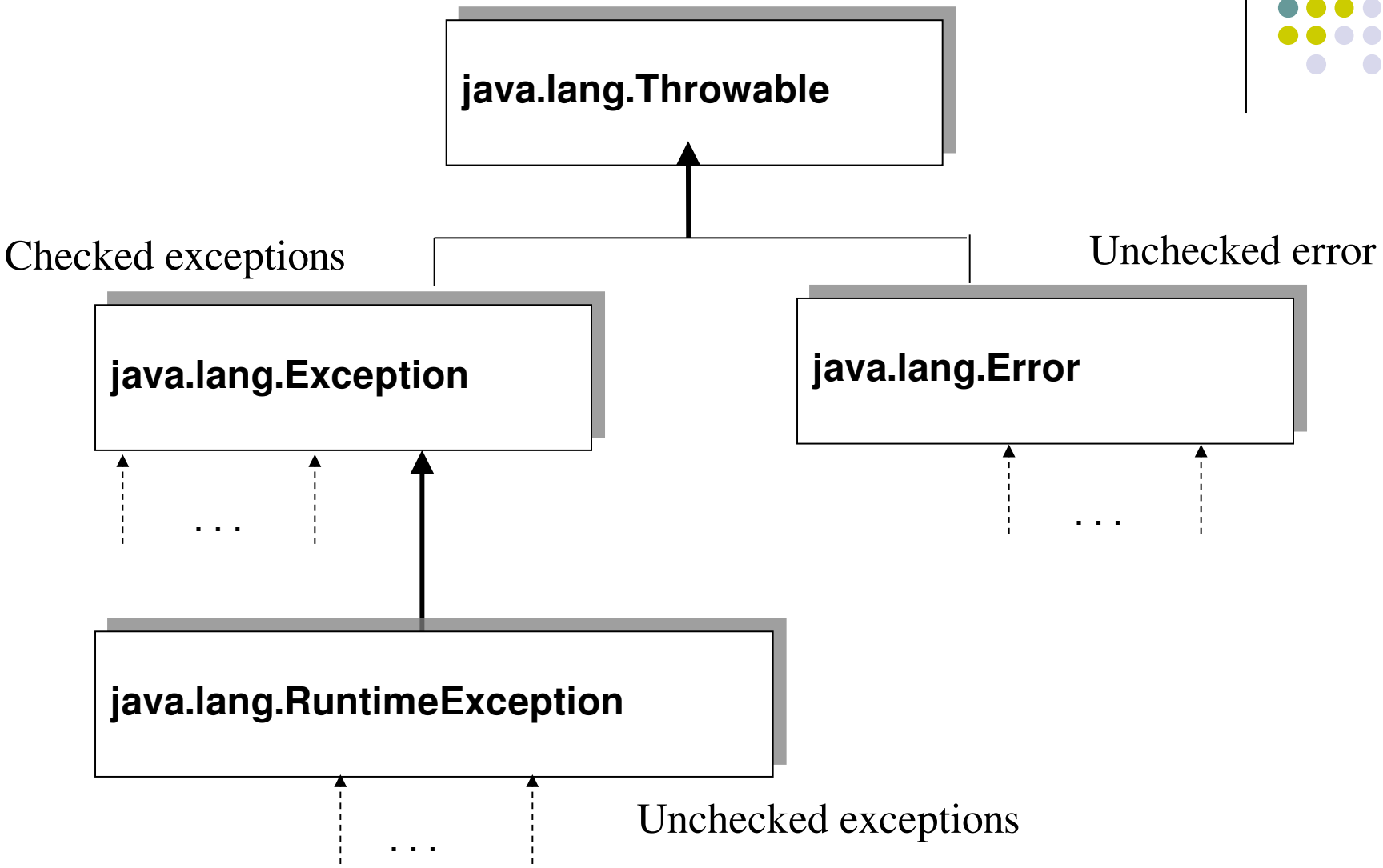
Esempio con blocco finally



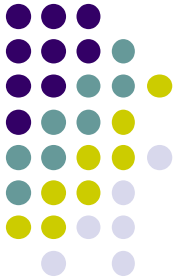
```
public class Esempio {
    // Metodo principale che viene eseguito all'avvio del programma
    public static void main(String[] args) {
        try {
            // Chiama il metodo 'metodo' che lancia un'eccezione di tipo 'MiaEccezione'
            metodo();
        } catch (MiaEccezione e) {
            // Cattura l'eccezione 'MiaEccezione' lanciata dal metodo
            // e stampa il messaggio dell'eccezione
            System.out.println("Eccezione catturata: " + e.getMessage());
        } finally {
            // Il blocco 'finally' viene sempre eseguito, indipendentemente
            // dal fatto che un'eccezione sia stata lanciata o meno
            System.out.println("Questo blocco viene sempre eseguito");
        }
    }

    // Metodo statico che lancia un'eccezione di tipo 'MiaEccezione'
    public static void metodo() throws MiaEccezione {
        // Lancia un'eccezione di tipo 'MiaEccezione' con un messaggio personalizzato
        throw new MiaEccezione("Errore personalizzato");
    }
}

// Classe personalizzata per definire un'eccezione specifica
class MiaEccezione extends Exception {
    // Costruttore della classe 'MiaEccezione' che accetta un messaggio di errore
    public MiaEccezione(String message) {
        super(message);
    }
}
```

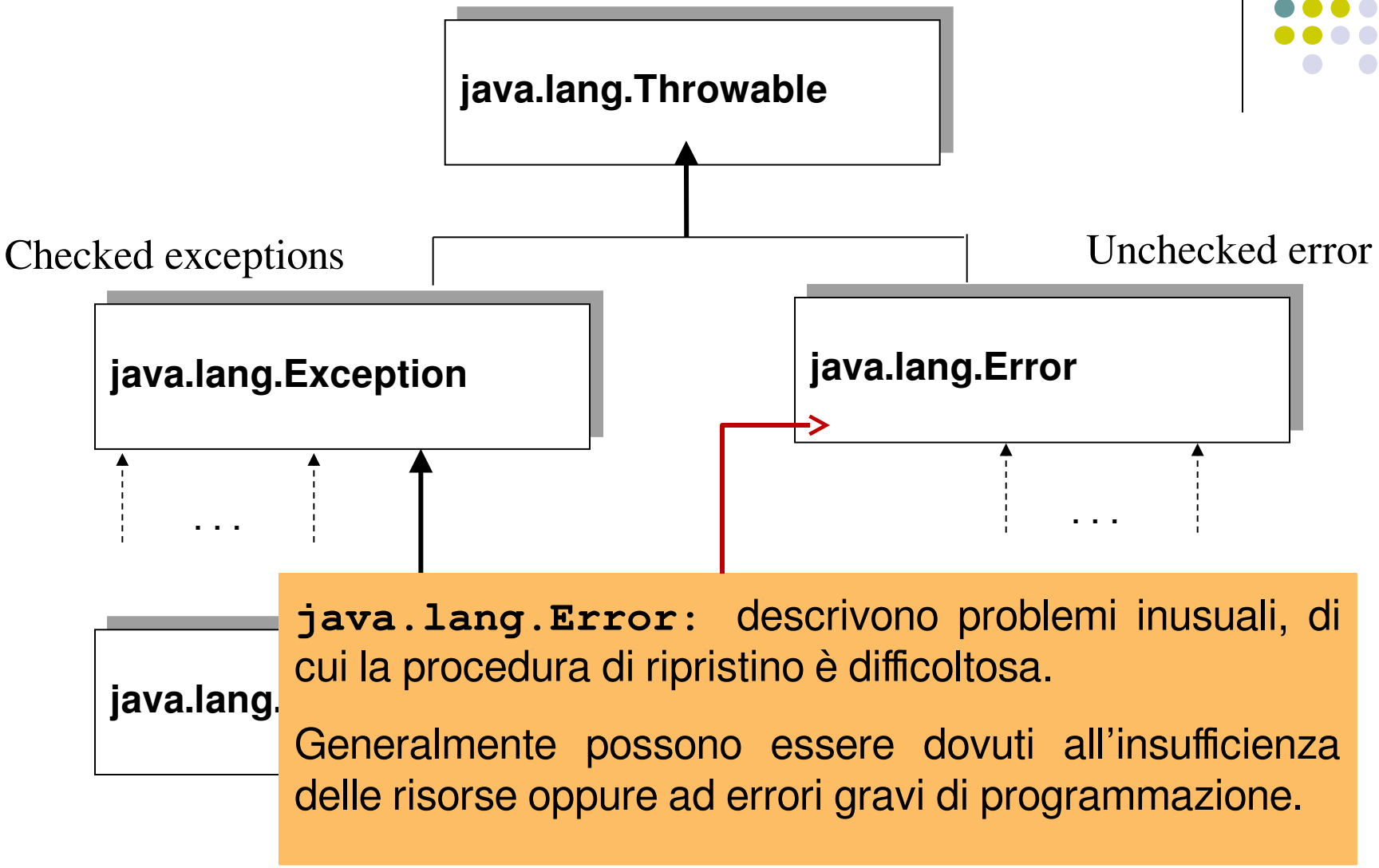
Classificazione delle eccezioni





Classificazione delle eccezioni

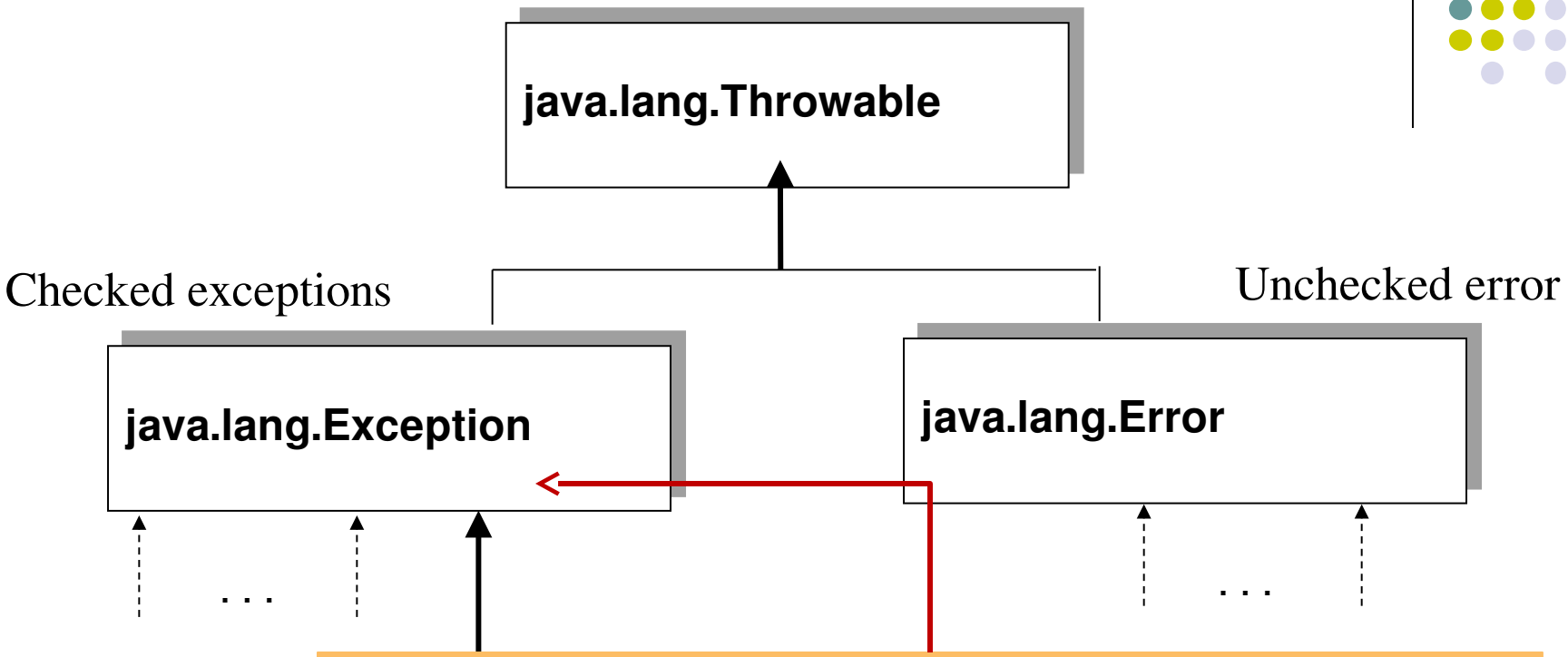




`java.lang.Error:` descrivono problemi inusuali, di cui la procedura di ripristino è difficoltosa.

Generalmente possono essere dovuti all'insufficienza delle risorse oppure ad errori gravi di programmazione.

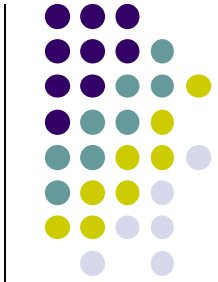
Classificazione delle eccezioni

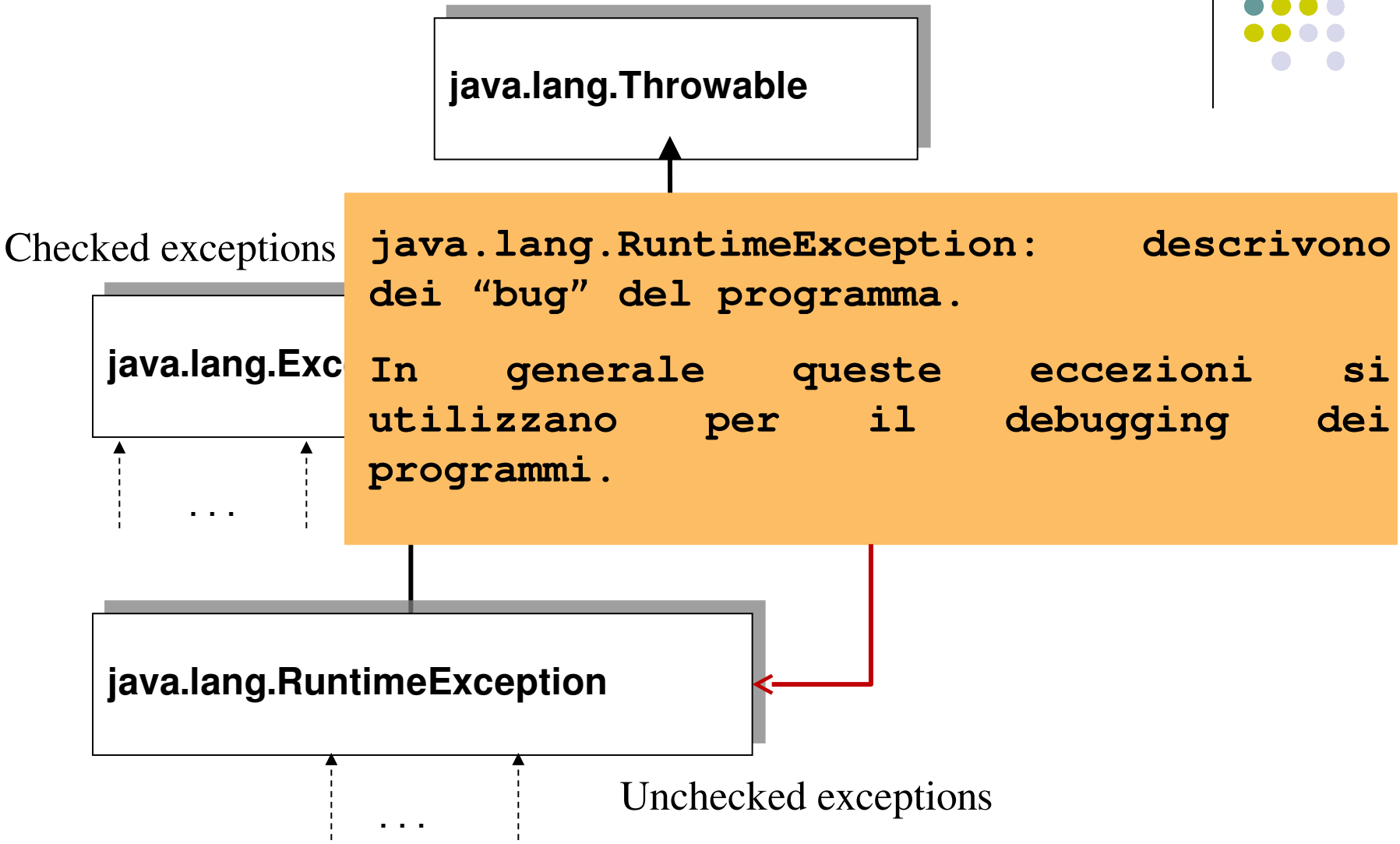


`java.lang.Exception:` descrivono problemi che si possono verificare nell'esecuzione di un'istruzione, dovuti ad imprevisti:

- problemi di I/O
- errori dovuti ad altri componenti

Classificazione delle eccezioni

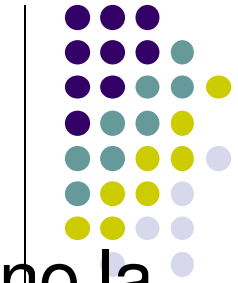




`java.lang.RuntimeException`: descrivono dei "bug" del programma.

In generale queste eccezioni si utilizzano per il debugging dei programmi.

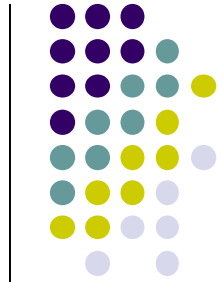
Dettagli sul funzionamento



La classe **Exception** e la classe **Error** estendono la classe Throwable per implementare il meccanismo con cui la Java Virtual Machine reagisce quando si imbatte in una eccezione-errore.

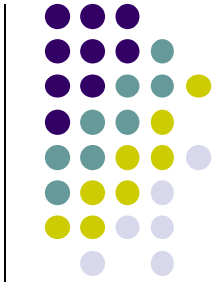
In caso di un'eccezione, la JVM istanzia un oggetto dalla classe eccezione relativa al problema (es. IOException), e lancia l'eccezione appena istanziata tramite la parola chiave throw.

Tipologie di eccezioni Java



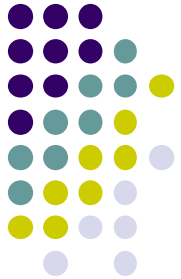
- **Unchecked Exceptions**
  Non è richiesto che questi tipi di eccezioni siano catturate o dichiarate in un metodo. Esempi sono:
  - *Runtime exceptions* the possono essere generate dai metodi o dalla JVM.
  - *Errors* che sono generati dall'interno della JVM e che spesso indicano un vero e proprio stato fatale dell'esecuzione
  - Le Runtime exceptions sono la sorgente delle maggiori controversie!
- **Checked Exceptions**
  Devono essere catturate da un metodo o dichiarate nella sua firma.
  - Aumentano la robustezza del codice a situazioni impreviste

Esempi



- IndexOutOfBoundsException :
  - Relativa a array, string, e vector
- ArrayStoreException :
  - Quando si assegna un oggetto di un tipo non corretto ad un elemento di un array
- NegativeArraySizeException :
  - Quando si usa una dimensione negativa di un array
- NullPointerException :
  - Quando si fa riferimento ad un oggetto non istanziato
- SecurityException :
  - Quando la security viene violata, sollevata dal security manager

Gestione delle eccezioni



Un'eccezione può essere gestita in due modalità:

- **diretta**: l'eccezione viene rilevata e "processata" nello stesso metodo;
- **indiretta**: l'esecuzione del metodo in cui è stata rilevata l'eccezione si arresta e l'eccezione viene "passata" al metodo chiamante.

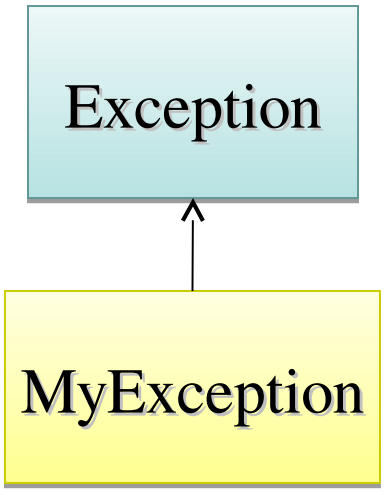
Eccezioni definite dall'utente

Il programmatore non è vincolato ad usare le eccezioni predefinite da Java, ma può crearne di proprie.

Per creare un tipo di eccezione d'utente, il programmatore deve derivarla da un tipo esistente, preferibilmente quella dal significato più vicino alla nuova eccezione.

```
import java.lang.Exception;

class MyException extends Exception {
public MyException() {}
public MyException(String msg) {
super(msg);
}
}
```

Eccezioni definite dall'utente



Consideriamo il diagramma delle classi a lato in cui un programmatore definisce una proprio eccezione a partire dalla classe Exception presente in `java.lang`.

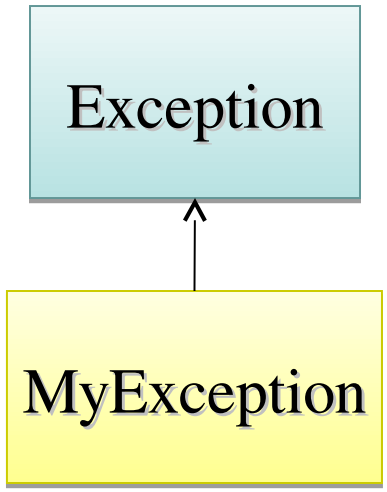
Supponiamo di avere il metodo

```
void do_smtg() throws MyException;
```

Consideriamo il seguente frammento di codice:

```
try{
…
do_smtg();
…
}catch(Exception ex) {…}
catch(MyException my_ex) {…}
```

Eccezioni definite dall'utente



Consideriamo il diagramma delle classi a lato in cui un programmatore definisce una proprio eccezione a partire dalla classe Exception presente in `java.lang`.

Suppon

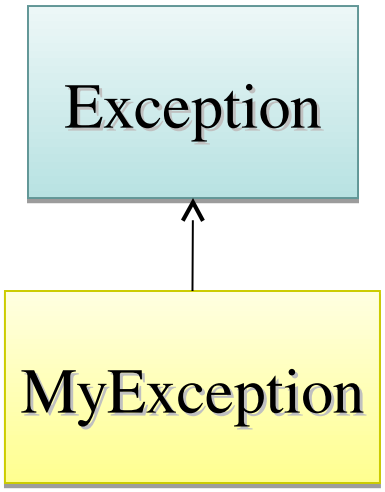
```
void do_
```

Consideriamo il segue

```
try{
…
do_smtg();
…
}catch(Exception ex) {…}
catch(MyException my_ex) {…}
```

Java tenta sempre di eseguire il primo blocco catch in grado di gestire l'eccezione appena sollevata. Il catch di Exception è utilizzabile per risolvere eccezioni di tipo MyException, quindi viene sempre eseguito, mentre l'altro mai.

Eccezioni definite dall'utente



Consideriamo il diagramma delle classi a lato in cui un programmatore definisce una proprio eccezione a partire dalla classe Exception presente in `java.lang`.

Supponiamo di avere il metodo

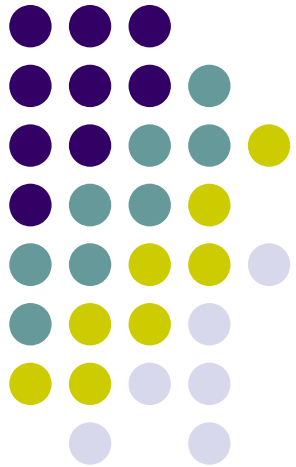
```
void do_smtg() throws MyException;
```

Consideriamo il seguente frammento di codice:

```
try{
…
do_smtg();
…
}catch(Exception ex) {…}
catch(MyException my_ex) {…}
```

Pertanto l'ordine dei blocchi catch deve essere l'inverso dell'ordine di derivazione: il catch di eccezioni derivate deve precedere quelli di eccezioni base.

Ricapitolando...



try, catch, finally

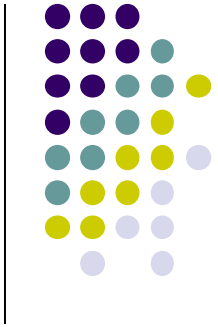
- La gestione delle eccezioni si basa sul meccanismo try–catch o attraverso, la clausola throws nella dichiarazione di un metodo.
  - `try { }`
  - `catch { }`

Nel blocco "try", viene inserita la sequenza di istruzioni che potenzialmente lancia un'eccezione.

```
try {
   …
   Istruzione;
   …
}
```

Se una eccezione è sollevata durante un blocco try,
il resto del codice
nel blocco try non è eseguito

try, catch, finally



- `try { }`
- `catch { }`

Nel blocco "catch", si inserisce il codice per il trattamento dell'eccezione (exception handler).

```
try {
   …
   Istruzione;
   …
} catch(TipoEccezione1 e1){
//trattamento eccezione 1
} catch(TipoEccezione2 e2){
//trattamento eccezione 2
}
```

try, catch, finally

Se un'eccezione non è intercettata dai blocchi "catch", essa sarà inoltrata al metodo chiamante.

```
try {
  …
  Istruzione;
  …
} catch(TipoEccezione1 e1){
//trattamento eccezione 1
} catch(TipoEccezione2 e2){
//trattamento eccezione 2
}
```

try, catch, finally

Se un'eccezione non è intercettata dai blocchi "catch", essa sarà inoltrata al metodo chiamante.

```
try {
   …
   Istruzione;
   …
} catch(TipoEccezione1 e1){
//trattamento eccezione 1
} catch(TipoEccezione2 e2){
//trattamento eccezione 2
}
```

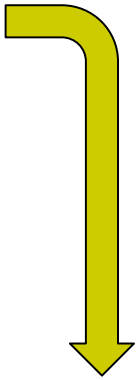
Un blocco try può essere seguito da una successione di blocchi catch: in quale ordine collocarli?

In base alla relazione gen/spec tra le Eccezioni: Dalla più specifica alla più generica

Multicatch (Java 7)

Da Java 7 in poi, un blocco catch può gestire più di un eccezione.

```
catch (IOException ex) {
     logger.log(ex);
     throw ex;
catch (SQLException ex) {
     logger.log(ex);
     throw ex;
}
```



Si riduce la duplicazione di codice e si evita la "tentazione" di fare il catch solo dell'eccezione più generale

```
catch (IOException|SQLException ex) {
     logger.log(ex);
     throw ex;
}
```

try, catch, finally

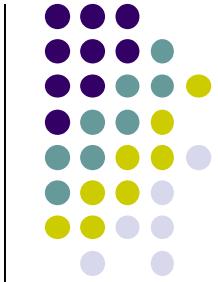
- Il flusso di controllo di un blocco try/catch non è predicibile,
  - dipende dalle eccezioni che vengono sollevate nel blocco `try;`
- Il blocco `finally` permette al programmatore di scrivere delle istruzioni che verranno eseguite incondizionatamente, a prescindere dal verificarsi di eccezioni;

try, catch, finally

- Nel caso venga sollevata un'eccezione, il blocco **"finally"** verrà eseguito dopo quello di **"try-catch"**.
- L'uso dei blocchi finally è quello di realizzare operazioni di clean-up.

```
try {
   istruzione1;
   ……
   istruzionen;
}catch ( TipoEccezione1 e1){
         //trattemento eccezione 1
} catch…

} finally {
         //codice che viene eseguito
         //in ogni caso
}
```

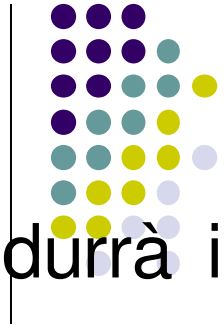
Esempio (1/3)



```
public class Ecc1 {
      public static void main(String args[]) {
      int a = 10;
      int b = 0;
      int c = a/b;
      System.out.println(c);
      }
}
```

•Questa classe può essere compilata senza problemi, ma genererà un'eccezione durante la sua esecuzione, dovuto alla divisione per zero.

Esempio (2/3)

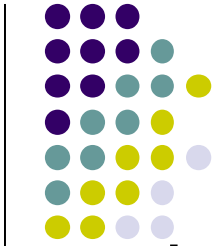


- La JVM dopo aver interrotto il programma produrrà il seguente output:

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
Ecc1.main(Ecc1.java:6)
```

- Evidenziando:
  - il tipo di eccezione (java.lang.ArithmeticException)
  - un messaggio descrittivo (/ by zero)
  - il metodo che ha sollevato l'eccezione (at Ecc1.main)
  - il file in cui è stata sollevata l'eccezione (Ecc1.java)
  - la riga in cui è stata sollevata l'eccezione (:6)

Esempio (3/3)

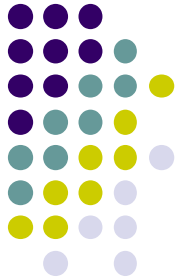


MA... il programma è terminato prematuramente!

Utilizzando le parole chiave try e catch sarà possibile gestire l'eccezione in maniera personalizzata:

```
public class Ecc2 {
   public static void main(String args[]) {
   int a = 10;
   int b = 0;
   try {
      int c = a/b;
         System.out.println(c);
      } catch (ArithmeticException exc) {
            System.out.println("Divisione per zero...");
            exc.printStackTrace();
            //stesso output di come se l'eccezione non fosse
            // catturata, ma senza interrompere il programma
            }
      }
}
```

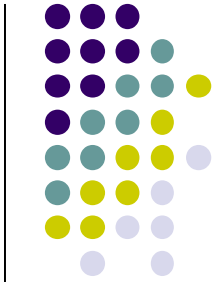
Throws



- In un blocco di codice che lancia un'eccezione si può decidere di servire l'eccezione (handling) o di rilanciare l'eccezione al chiamante
  - Per servire l'eccezione si scrive un blocco try-catch.
  - Per passare l'eccezione al chiamante, si inserisce la clausola throws nella dichiarazione del metodo

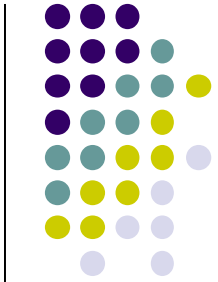
```
public void myMethod() throws IOException {
                //… codice con operazioni di I/O
  }
```

Riferimenti



- Programmare in Java
  - Cap. 11 fino al §11.4, stack unwinding §11.17

Conclusioni



- Le interruzioni sono gestite tramite eccezioni.
- Le librerie Java forniscono vari strumenti per la gestione delle eccezioni.
- È possibile definire eccezioni personalizzate per casi specifici.

- Vantaggi delle eccezioni:
  - Migliora la gestione degli errori.
  - Rende il codice più robusto e leggibile.