Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Dipartimento di Elettronica e Informazione

Esercitazioni di Ingegneria del Software

05: Generics, Eccezioni, Ordinamento

Liste & Generics

- Relazioni tra oggetti:
  - Object o = new Object();
  - Integer i = new Integer(5);
  - Object ok = i;
- Relazioni tra liste:
  - List<Object> lo = new ArrayList<Object>();
  - List<Integer> li = new ArrayList<Integer>();
  - ~~List<Object> err = new ArrayList<Integer>();~~
- … ma perché?

Liste & Generics (cont).

- Esercizio: Lo Zoo (tratto dal Java tutorial)

- Supponiamo di avere degli animali:
  - `abstract class Animale { ... }`
  - `class Leone extends Animale { ... }`
  - `class Libellula extends Animale { ... }`
- Possiamo avere:
  - `Leone leo = new Leone();`
  - `Libellula lib = new Libellula();`

  - `Animale a1 = leo;`
  - `Animale a2 = lib;`

Liste & Generics (cont).

- Supponiamo di avere delle gabbie Generiche:
  - `class Gabbia<T> { ... }`
- Possiamo creare:
  - Gabbie resistenti per leoni:

    ```
    Gabbia<Leone> resistente = new Gabbia<Leone>();
    resistente.add(leone);
    ```
  - Gabbie fitte per libellule:

    ```
    Gabbia<Libellula> fitta = new Gabbia<Libellula>();
    fitta.add(libellula);
    ```
  - Gabbie per qualsiasi animale (resistenti, fitte, impermeabili, …):

    ```
    Gabbia<Animale> flessibile = new Gabbia<Animale>();
    flessibile.add(leone);
    flessibile.add(libellula);
    ```

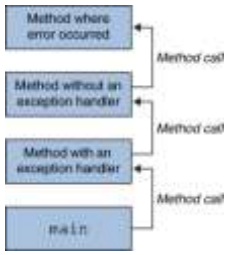
Liste & Generics (cont).

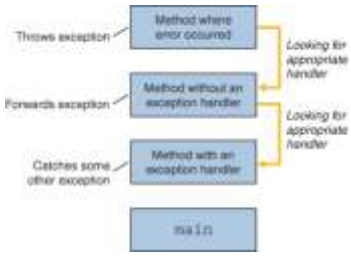
- Consideriamo le diverse gabbie:
  - Se leone è un tipo animale,
    una gabbia per leoni è un tipo di gabbie per animali?
    - Dovrebbe essere
      - Resistente
      - Fitta
      - Impermeabile
      - …

```
Gabbia<Animale> flessibile = new Gabbia<Leone>();
```


Eccezioni: teoria

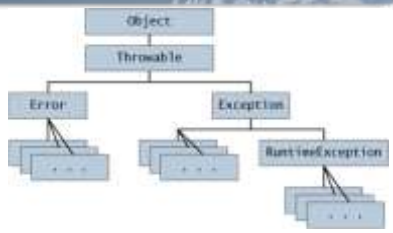
- **Definition:** An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.





Eccezioni: teoria

- Tipi di eccezioni
  - Checked:
    - devono essere gestite
      - » O catchate
      - » O rilanciate esplicitamente
  - *Unchecked*
    - Error
      - rappresentano situazioni eccezionali esterne all'applicazione, che è difficile prevedere e risolvere.
      - Sono sottoclassi di Error
      - Non devono essere gestite necessariamente
    - Runtime Exceptions
      - rappresentano situazioni eccezionali, interne all'applicazione, che difficilmente si possono prevedere o risolvere
      - Sono sottoclassi di RuntimeException
      - Non devono essere necessariamente gestite



Eccezioni unchecked: esercizi

- Cosa stampa?

```
public class Launcher {
    public static void main(String[] args) {
        float media = calcolaMedia(args);
        System.out.println("La media è: " + media);
    }

    private static float calcolaMedia(String[] args) {
        int tot = 0;
        int num = 0;

        for (String arg : args) {
            int i = Integer.parseInt(arg);
            tot += i;
            num++;
        }

        return tot / num;
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Launcher.calcolaMedia(Launcher.java:17)
        at Launcher.main(Launcher.java:3)
```

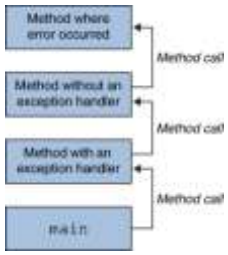
Eccezioni unchecked: esercizi

```
public class Launcher {
   public static void main(String[] args) {
       String num = args[0];
       System.out.println("Il numero è " + num);
   }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at Launcher.main(Launcher.java:4)
```


Eccezioni

- Cos'è successo di sbagliato che ha portato all'errore?




```
dire.registry.exception.AuthenticationException
   at dire.registry.Registry.auth(Registry.java:49)
   at dire.registry.Registry.authenticate(Registry.java:44)
   at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
   at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:79)
   at java.lang.reflect.Method.invoke(Method.java:618)
   at org.jboss.aop.joinpoint.MethodInvocation.invokeNext(MethodInvocation.java:112)
   at org.jboss.ejb3.interceptor.EJB3InterceptorsInterceptor.invoke
   at org.jboss.aop.joinpoint.MethodInvocation.invokeNext(MethodInvocation.java:101)
   at org.jboss.aop.joinpoint.MethodInvocation.invokeNext(MethodInvocation.java:101)
   at org.jboss.aop.joinpoint.MethodInvocation.invokeNext(MethodInvocation.java:101)
   at org.jboss.aspects.tx.TxPolicy.invokeInOurTx(TxPolicy.java:79)
   at org.jboss.aspects.tx.TxInterceptor$Required.invoke(TxInterceptor.java:191)
   at org.jboss.aop.joinpoint.MethodInvocation.invokeNext(MethodInvocation.java:101)
   at org.jboss.aspects.tx.TxPropagationInterceptor.invoke
   at org.jboss.aop.joinpoint.MethodInvocation.invokeNext(MethodInvocation.java:101)
```

Eccezioni: gestiamole!

```
public void writeList(int[] values) {
   PrintWriter out = null;

   try {
        out = new PrintWriter(new FileWriter("OutFile.txt"));

        for (int i = 0; i < values.length-1; i++) {
               float avg = values[i] / values[i+1];
               out.println("Ratio: " + avg);
        }

   } catch (ArithmeticException e) {
        System.err.println("Non voglio zeri! " + e.getMessage());
   } catch (IOException e) {
        System.err.println("Errore di scritt: " + e.getMessage());
   } finally {
        if (out != null) out.close();
   }
}
```


Progettare le Eccezioni

- La presenza di zeri nell'array, può essere gestito meglio!

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

Eccezioni: gestiamole! (2)

```
public void writeList(int[] values) throws CiSonoZeriException {
   PrintWriter out = null;

   try {
        out = new PrintWriter(new FileWriter("OutFile.txt"));

        for (int i = 0; i < values.length-1; i++) {
                float avg = values[i] / values[i+1];
                out.println("Ratio: " + avg);
        }

   } catch (ArithmeticException e) {
        throw new CiSonoZeriException("L'array deve essere > 0");
   } catch (IOException e) {
        System.er
   } finally {
        if (out !
   }
}
```

```
int[] values = // leggili dall'utente
try {
  writeList(values);
} catch (CiSonoZeriException e) {
  // notifica l'utente
}
```

Eccezioni: un esercizio

```
class EccPiccolo extends Exception {
    public EccPiccolo(String s) {super(s);}
}

class EccGrande extends Exception {
    public EccGrande(String s) {super(s);}
}

class EccGrandeGrande extends EccGrande {
    public EccGrandeGrande(String s) {super(s);}
}
```

Eccezioni: un esercizio

```
static void m1() throws EccPiccolo {
  throw new EccPiccolo("m1");
}

static void m2() throws EccGrandeGrande {
  try {
    m1();
  } catch (EccPiccolo e) {
    throw new EccGrandeGrande("m2: " + e.getMessage());
  } }

static void m3() throws EccGrande {
  m2();
}

public static void main (String[] args) {
  try {
    m3();
  } catch (EccGrande e) {
    e.printStackTrace();
  } } }
```

```
EccGrandeGrande: m2: m1
    at Ecc.m2(Ecc.java:27)
    at Ecc.m3(Ecc.java:32)
    at Ecc.main(Ecc.java:37)
```


Ordinamento

- Per confrontare gli oggetti, in Java esiste l'interfaccia:

```
public interface Comparable<T> {
   public int compareTo(T arg0);
}
```

- Possiamo definire come si ordinano le persone:

```
public class Persona implements Comparable<Persona> {
   private String nome, cognome;
   private int eta;

   public Persona(String nome, String cognome, int eta) { … }

   // …

   @Override
  public int compareTo(Persona arg0) {
    if (this.cognome.compareTo(arg0.cognome) != 0)
      return this.cognome.compareTo(arg0.cognome);

    return this.nome.compareTo(arg0.nome);
  } }
```

Ordinamento

- Possiamo quindi ordinare facilmente un elenco di persone:

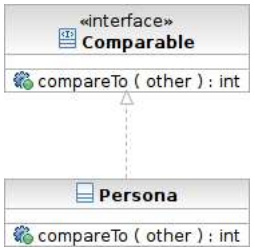
```
public class Launcher {
  public static void main(String args[]) {
    Persona[] elenco = new Persona[]{
      new Persona("Matteo","Miraz", 1),
      new Persona("Luciano","Baresi", 2),
      new Persona("Ultimo","ZZZ", 0)
    };

    Arrays.sort(elenco, 0, elenco.length);

    for (Persona p : elenco) System.out.println(p);
  }
}
```


Ordinamento

- Osservazioni
  - La classe diventa in grado di confrontarsi e stabilire la relazione di ordinamento



- Problema
  - Cosa succede se vogliamo ordinare classi di altri, che non implementano l'interfaccia Comparable ?
  - Cosa succede se non esiste un unico modo per ordinare i dati?

Ordinamento

- È possibile spostare la capacità di confrontare elementi in una classe apposta
  - I dati rimangono classi “normali”
  - Sono create altre classi in grado di confrontare i dati

Ordinamento

```
public class OrdinaPerNome implements Comparator<Persona> {
  public int compare(Persona p1, Persona p2) {
    if(p1.getNome().compareTo(p2.getNome()) != 0)
      return p1.getNome().compareTo(p2.getNome());
    else
      return p1.getCognome().compareTo(p2.getCognome());
  }
}
```

```
public class OrdinaPerCognome implements Comparator<Persona> {
  public int compare(Persona p1, Persona p2) {
    if(p1.getCognome().compareTo(p2.getCognome()) != 0)
      return p1.getCognome().compareTo(p2.getCognome());
    else
      return p1.getNome().compareTo(p2.getNome());
  }
}
```

```
public class OrdinaPerEta implements Comparator<Persona> {
  public int compare(Persona p1, Persona p2) {
    return p1.getEta() - p2.getEta();
  }
}
```

Ordinamento

```
public static void main(String[] args) {
  Persona[] persone = new Persona[] {
    new Persona("Mario", "Rossi", 30),
    new Persona("Giuseppe", "Verdi", 25),
    new Persona("Carlo", "Bianchi", 18)
  };

  for (Persona persona : persone)
    System.out.println(persona);

  System.out.println("-- ordino per nome ---");
  Arrays.sort(persone, new OrdinaPerNome());
  for (Persona persona : persone)
    System.out.println(persona);

  System.out.println("-- ordino per cognome ---");
  Arrays.sort(persone, new OrdinaPerCognome());
  for (Persona persona : persone)
    System.out.println(persona);

  System.out.println("-- ordino per eta ---");
  Arrays.sort(persone, new OrdinaPerEta());
  for (Persona persona : persone)
    System.out.println(persona);
}
```

```
Rossi Mario 30
Verdi Giuseppe 25
Bianchi Carlo 18
-- ordino per nome ---
Bianchi Carlo 18
Verdi Giuseppe 25
Rossi Mario 30
-- ordino per cognome ---
Bianchi Carlo 18
Rossi Mario 30
Verdi Giuseppe 25
-- ordino per eta ---
Bianchi Carlo 18
Verdi Giuseppe 25
Rossi Mario 30
```