Classi di Java introducono Oggetti e Tipi per il programma

- Un numero finito di *entità statiche*.
- Un numero infinito di *entità dinamiche*: **Oggetti**.

```
import java.io.*;
import java.util.*;

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

  - Gli oggetti sono valori e sono usati come sotto

```
public class ACUse {
/* Classe ACUse mostra un uso degli oggetti di tipi AbstractCounter */
//metodi
    public static void main (String [] args) {
        AbstractCounter myCounterA, myCounterB; //due variabili di tipo...
        myCounterB = new AbstractCounter();  //crea un oggetto di tipo ... e lo assegna a...
        myCounterB.inc();
        myCounterA = myCounterB;
        myCounterA.inc();
        System.out.println("myCounterA segna "+myCounterA.get());
        System.out.println("myCounterB segna "+myCounterB.get());
    }
}
```

  - che rappresenta il modo standard di uso delle classi di Java

Gli Oggetti sono valori con modello "value reference"

- Cosa viene stampato?.

```
import java.io.*;
import java.util.*;

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

```
public class ACUse {
/* Classe ACUse mostra un uso degli oggetti di tipi AbstractCounter */
//metodi
    public static void main (String [] args) {
        AbstractCounter myCounterA, myCounterB; //due variabili di tipo...
        myCounterB = new AbstractCounter();  //crea un oggetto di tipo ... e lo assegna a...
        myCounterB.inc();
        myCounterA = myCounterB;
        myCounterA.inc();
        System.out.println("myCounterA segna "+myCounterA.get());
        System.out.println("myCounterB segna "+myCounterB.get());
    }
}
```

- Gli Oggetti in Java, sono:

  valori reference = il valore coincide con un reference alla struttura di memoria occupata.

Struttura dei Programmi: Relazioni tra Classi

I programmi sono una collezione strutturata di Classi (con propria funzionalità)

- La struttura della collezione è definita dalla **Relazione tra Classi**
- In Java la relazione è esplicita e ha due forme.
  - **extends**. Classe-Sottoclasse. Ereditarità singola
  - **implements**. Interfaccia-Sottoclasse. Ereditarità multipla
- La struttura è fondamentale per realizzare funzionalità con comportamento via, via, più complicato
- Parte centrale della programmazione OO in Java.

Principi: Funzionalità Data-Centric

I programmi sono una collezione strutturata di Classi (con propria funzionalità)

- La funzionalità è espressa da 1 o più Classi (in Java) correlate.
- Le Classi usualmente introducono Oggetti (con un Tipo).
- La funzionalità allora è espressa in forme simili a quella sotto

```
import java.io.*;
import java.util.*;

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

- **Data-Centric** dove vediamo una classe di valori importati per il problema da risolvere e le operazioni che possiamo usare su essi per risolvere il problema o una sua parte (metodologie di programmazione Data-Centric)

Principi: Riuso di Codice

Quando la realizzazione di un sistema A è stata completata e la sua esecuzione è a regime, la scrittura del codice di A entra in una nuova fase: La manutenzione.

- È una fase di programmazione molto impegnativa e costosa.
- **Riuso** significa contenere il codice che deve essere prodotto e gli errori conseguenti
- Nell'esempio sotto, A è esteso con una nuova classe di contatori che riusano il codice dei contatori già presenti, ma

```
public class NewCounter extends AbstractCounter{
/* Classe NewCounter per contatori AbstractCounter con operazione
   alive che indica quanti NewCounter sono stati prodotti.
*/
    private static int counterCount = 0;
    public static int alive(){return counterCount;}
//metodi
    public NewCounter(){counterCount++;}
}
```

- sono in grado di dire quanti contatori sono in uso.

Principi: Riuso di Codice/2

- A è esteso con una nuova classe di contatori che riusano il codice dei contatori già presenti, ma hanno delle proprietà aggiuntive

```
public class NewCounter extends AbstractCounter{
/* Classe NewCounter per contatori AbstractCounter con operazione
   alive che indica quanti NewCounter sono stati prodotti.
*/
    private static int counterCount = 0;
    public static int alive(){return counterCount;}
//metodi
    public NewCounter(){counterCount++;}
}
```

- vecchie e nuove parti di A convivono perfettamente.

```
public class ACUse {
/* Classe ACUse mostra un uso degli oggetti di tipi AbstractCounter e NewCounter */
//metodi
    public static void main (String [] args) {
        AbstractCounter myCounterA, myCounterB; //due variabili di tipo...
        myCounterB = new AbstractCounter();  //crea un oggetto di tipo ... e lo assegna a...
        myCounterB.inc();
        myCounterA = myCounterB;
        myCounterA.inc();
        System.out.println("myCounterA segna "+myCounterA.get());
        System.out.println("myCounterB segna "+myCounterB.get());
        {//uso di NewCounter
            NewCounter x;
            for (int i=0; i+myCounterA.get()<10; i++) {//i due tipi di contatore usati insiemee New
                NewCounter y = new NewCounter();
                x=y;
                x.inc();
            }
        }
        System.out.println("sono stati usati "+NewCounter.alive()+" NewCounter");
    }
}
```

Principi: ADT, Moduli, Oggetti, Packages, API

- ADT. Non presenti in Java come meccanismi specifici, ma
  - Java ha meccanismi **Modificatori** che:
  - regolano la visibilità di Classi, Oggetti, campi e metodi e
  - forniscono Classi in grado di comportarsi come ADT

```
import java.io.*;
import java.util.*;

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

- Moduli. Classi e Oggetti in Java sono Unità di Programmazione[5], ma per la programmazione in grande
  - Java combina **Modificatori** con **Packages** e **Applicative Programming Interface**:

[5] queste unità, in Java, possono essere viste come moduli per programmazione in piccolo

Principi: Applicazioni Multi-Media, Multi-Contesto

- **Multi-Media**. La modularità e l'astrazione racchiuse negli Oggetti permette di **integrare strutture** di natura molto diversa tra loro quali suoni, immagini, video.
- **Multi-Contesto**. Il modello Data-Centric si presta a programmare applicazioni in contesti molto diversi quali commerciali, amministrativi, gestionali e scientifici.
- **Verbosità e Efficienza**. Sono i veri limiti dell'approccio OO.

Java: Guida all'istallazione e Uso

La nostra piattaforma (Linux, Mac, Windows, ...) deve avere istallato il Java Development Kit: jdk1.6.XX o versioni superiori

- **Download e Istallazione**.
  (a) Connettersi al sito http://www.oracle.com/technetwork/java/javaee/downloads/index.html
  (b) selezionare la versione più avanzata per la propria macchina
  (c) istallare il jdk scaricato, seguendo le indicazioni
- **Selezione Directories**.
  (a) Directory jdkXXX/bin dei comandi javac (compilatore) e java (JVM).
  individuare il cammino PTool alla directory "jdkXXX/bin"
  (b) Directory YYYY da usare per i propri programmi Java.
  creare una directory YYYY e e individuare il cammino PProgram ad essa
- **Setting delle variabili di ambiente**.
  Variabili coinvolte:
  path = PTool
  classpath = PProgram
  Per le diverse piattaforme vedere[6]
  https://docs.oracle.com/javase/tutorial/essential/io/pathClass.html
- **Uso**.
  (a) Eding del programma:
  classi del programma hanno nome con suffisso ".java", sono editate con un text editor,
  poste in una directory XX appositamente creata e posta nella directory PT.
  (b) Compilazione del programma: javac XX/c1.java ... XX/ck.java
  eseguire in command prompt (Windows), Terminal (Mac)
  produce k file ci.class contenenti il bytecode della corrispondente classe
  (c) Esecuzione: java XX/cm
  esegue il codice del metodo main presente nella classe cm.class.

[6] per Mac, operare sempre da Terminal seguendo le indicazioni per Linux.