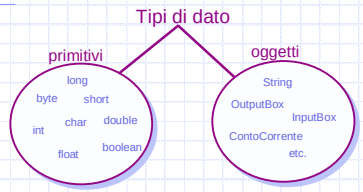
Approfondimenti su classi e oggetti

Unità didattica 6
Armando Tacchella
Fondamenti di Informatica

Tipi primitivi e oggetti



- Vi sono due tipi di dato: primitivi e oggetti
- I tipi di dato (non numerici) char e boolean e tutti i tipi di dato numerici sono primitivi
- Gli oggetti sono trattati in maniera diversa in quanto la loro occupazione di memoria non è predeterminata

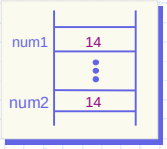
Assegnamento con tipi primitivi

Sorgente

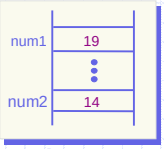
A
```
int num1, num2;
num1 = 14;
num2 = num1;
num1 += 5;
```

B
```
int num1, num2;
num1 = 14;
num2 = num1;
num1 += 5;
```

Memoria



Dopo A



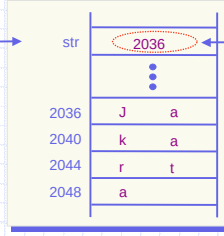
Dopo B

Allocazione di memoria per oggetti

Sorgente

```
String str;
str = "Jakarta";
```

str è una variabile String, (un tipo reference) il cui contenuto è un indirizzo di memoria (reference).



Il valore 2036 è l'indirizzo di memoria a partire da cui la stringa è contenuta.

Assumendo quattro byte per ogni riga, vi sono due caratteri per riga (16 bit per ogni char).

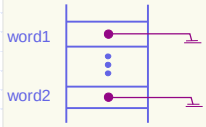
Memoria

Assegnamenti tra oggetti (1)

Sorgente

A
```
String  word1, word2;
word1 = new String( "Java" );
word2 = word1;
```

Sia word1 che word2 sono allocate in memoria (contengono gli indirizzi) ma gli oggetti non sono ancora creati, così entrambi sono null.



Memoria

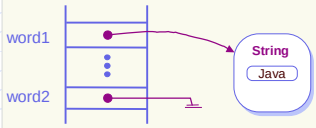
Dopo A

Assegnamenti tra oggetti (2)

Sorgente

B
```
String  word1, word2;
word1 = new String( "Java" );
word2 = word1;
```

Un oggetto String viene creato e assegnato a word1, quindi word1 contiene l'indirizzo di questo oggetto.



Memoria

Dopo B

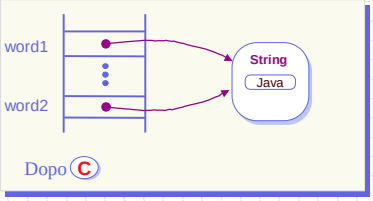
Assegnamenti tra oggetti (3)

Sorgente

```
String  word1, word2;
word1 = new String( "Java" );
word2 = word1;
```

C

Il contenuto di word1, che è un indirizzo, è assegnato a word2, e quindi word2 si riferisce allo stesso oggetto di word1.

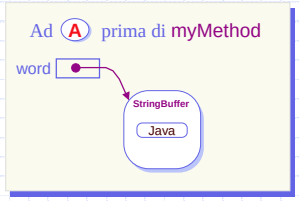


Passaggio di oggetti a metodi (1)

Sorgente A

```
// word è stato già creato
tester.m( word );
```

```
public void m( StringBuffer strBuf ) {
    strBuf.setCharAt( 0, 'Y' );
}
```



A. Le variabili locali non esistono prima dell'esecuzione del metodo.

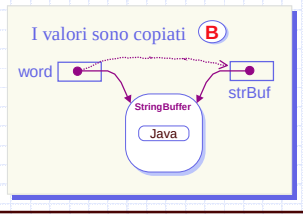
Passaggio di oggetti a metodi (2)

Sorgente

```
// word è stato già creato
tester.m( word );
```

```
public void m( StringBuffer strBuf ) {
    strBuf.setCharAt( 0, 'Y' );
}
```

B



B. Il valore dell'argomento, che è un indirizzo, è copiato nel parametro.

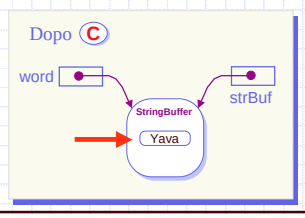
Passaggio di oggetti a metodi (3)

Sorgente

```
// word è stato già creato
tester.m( word );
```

```
public void m( StringBuffer strBuf ) {
    strBuf.setCharAt( 0, 'Y' );
}
```

C



C. Il contenuto dell'oggetto indirizzato da strBuf è cambiato.

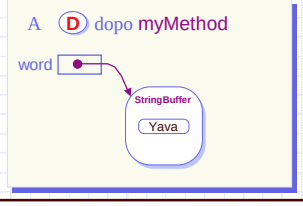
Passaggio di oggetti a metodi (4)

Sorgente

```
// word è stato già creato
tester.m( word );
```

D

```
public void m( StringBuffer strBuf ) {
    strBuf.setCharAt( 0, 'Y' );
}
```



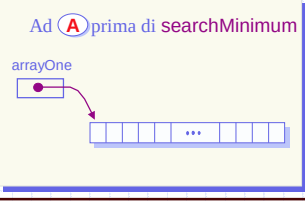
D. Il parametro è deallocato. L'argomento punta ancora allo stesso oggetto (modificato).

Passaggio di vettori ai metodi (1)

Sorgente A

```
minOne
= searchMinimum(arrayOne);
```

```
public int searchMinimum(float[] number))
{
    …
}
```



A. La variabile locale number non esiste prima dell'esecuzione del metodo.

Passaggio di vettori ai metodi (2)

Sorgente

```
minOne
= searchMinimum(arrayOne);
```

```
public int searchMinimum(float[] number))
{
    …
}
```

B

Memoria

L'indirizzo è copiato in B

arrayOne

number

B. Il valore dell'argomento (un indirizzo), viene copiato nel parametro.

Passaggio di vettori ai metodi (3)

Sorgente

```
minOne
= searchMinimum(arrayOne);
```

```
public int searchMinimum(float[] number))
{
    …
}
```

C

Memoria

Durante C dentro il metodo

arrayOne

number

C. L'accesso al vettore avviene tramite number all'interno del metodo.

Passaggio di vettori ai metodi (4)

Sorgenti

```
minOne
= searchMinimum(arrayOne);
```

D

```
public int searchMinimum(float[] number))
{
    …
}
```

Memoria

A D dopo searchMinimum

arrayOne

D. Il parametro è deallocato. L'argomento punta ancora allo stesso oggetto.

Restituzione di oggetti da metodi

- "Passare un oggetto ad un metodo come argomento" significa passare per valore il riferimento dell'oggetto
- "Restituire un ogetto da un metodo" significa che l'oggetto è creato all'interno del metodo e restituito tramite l'istruzione return
- Ad esempio, per convertire l'intero x in una stringa:

```
public String trasforma( int x ) {
    String xs = String.valueOf(x);
    return xs;
}
```

- L'oggetto corrispondente a xs è creato una sola volta; ad essere restituito per valore è il riferimento all'oggetto

Puntatore di auto-riferimento

- La parola chiave this è utilizzata per riferirsi agli attributi e ai metodi di un oggetto (nonchè all'oggetto stesso)
- La parola chiave this non è necessaria se non vi sono possibili ambiguità.

```
class Tester {
    public void m1( ){
        ...
    }
    public void m2( ) {
        m1( );
    }
}
```

```
class Tester {
    public void m1( ){
        ...
    }
    public void m2( ){
        this.m1( );
    }
}
```

Utilizzo del puntatore di auto-riferimento (1)

- A cosa si riferisce l'identificatore age?

```
class Person
{
  int age;
  ...
  public void setAge( int age )
  {
    ... age ...

  }
}
```

C'è un corrispondente parametro locale e quindi age si riferisce al parametro.

```
class Person
{
    int age;
    ...
    public void setAge( int pAge )
    {
      ... age ...

    }
}
```

Non c'è un corrispondente parametro locale e quindi age si riferisce all'attributo.

Utilizzo del puntatore di auto-riferimento (2)

```
class Person
{
  int age;
  ...
  public void setAge( int age )
  {
    this.age  =  age ;

  }
}
```

Utilizzo del puntatore di auto-riferimento (3)

- La parola chiave this può essere utilizzata per chiamare un costruttore all'interno di un altro costruttore della stessa classe.

```
class Person
{
  public void Person( )
  {
    this( "Not Given", 0, 'U' );
  }
  public void Person( String name, int age, char gender )
  {
     this.name = name;
     this.age  = age;
     this.gender = gender;
  }
     …
}
```

Questa classe ha due costruttori. Il primo costruttore chiama il secondo con valori di default per gli argomenti.