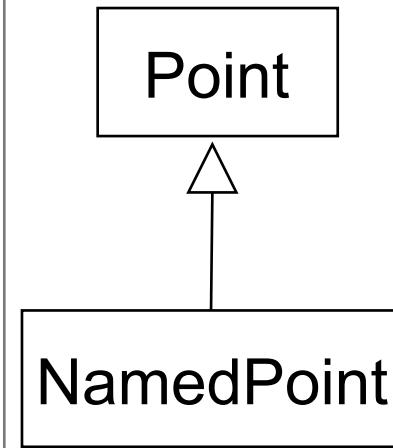




Determinazione del tipo



```
public static void main(String a[]){
    Point p;
    // leggi k
    if (k==1) p=new Point(2,2);
    else p=new NamedPoint(3,3,"A");

    // p.getName(); SBAGLIATO!

    if (p instanceof NamedPoint)
        ((NamedPoint)p).getName();
}
```



Static and Dynamic binding

Cosa succede quando si chiama un metodo su di un oggetto
(esempio: C obj; ... obj.f(args)) ?

- 1.il compilatore cerca tra i metodi dell'oggetto obj i metodi f(...) e li enumera
- 2.il compilatore determina i tipi di parametri passati. Se trova un match unico con i metodi della classe C (anche tramite cast!) ne prende nota – **overloading resolution** – altrimenti genera un messaggio di errore
- 3.se il metodo è private, static, final o un costruttore il compilatore sa esattamente che metodo chiamare (**static binding**)
- 4.altrimenti il metodo dipende da qual'è la classe a cui obj appartiene a runtime ed il compilatore deve delegare alla JVM la determinazione del metodo a run time (**dynamic binding**)



Static and Dynamic binding

Se il programma usa il dynamic binding, la JVM deve chiamare il metodo appropriato al particolare tipo di oggetto obj.

- Es. **obj** sia di tipo D, classe derivata da C. Se esiste un metodo **f(int)** in D, questo sarà il metodo chiamato, altrimenti verrà chiamato il metodo **f(int)** di C
- Eseguire ogni volta questi controlli non è efficiente., per cui la virtual machine calcola in anticipo un method table per ogni classe che raccoglie tutte le signatures dei metodi



Polimorfismo - esempio

```
abstract class OP {  
    int f(int a,int b);  
}  
  
class Somma extends OP {  
    int f(int a,int b){  
        return a+b;  
    }  
}  
  
class Sottrazione extends OP {  
    int f(int a,int b){  
        return a-b;  
    }  
}
```



Polimorfismo - esempio

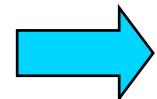
```
class Test {  
    public static void main(String[] a) {  
        new Test();  
    }  
    public Test() {  
        OP o;  
        ...  
        if (i!=0) o=new Somma();  
        else o=new Sottrazione();  
        ...  
        o.f(3,2);  
    }  
}
```

Che metodo viene chiamato qui?



Classi Wrapper

classe "wrapper"



<code>int</code>	<code>Integer</code>
<code>float</code>	<code>Float</code>
<code>char</code>	<code>Char</code>
<code>byte</code>	<code>Byte</code>
<code>double</code>	<code>Double</code>

Generano **oggetti** che hanno come **variabile di istanza**
un tipo di dato primitivo con nome uguale

Integer

int

Float

float

Sezione: Pila Polimorfa - richiami



Una Pila Polimorfa



La Pila in Java - 1

```
package strutture;
public class Pila {
    int size;
    int marker;
    final int DEFAULTGROWTHSIZE=5;
    Object contenuto[];
    final int initialSize=3;

    Pila() {
        size=initialSize;
        marker=0;
        contenuto=new Object[size];
    }
}
```



La Pila in Java - 2

Abilità lo
static
binding

```
final void inserisci(Object k) {  
    if (marker==size)  
        {cresci(DEFAULTGROWTHSIZE);}  
    contenuto[marker]=k;  
    marker++;  
}  
  
Object estrai() {  
    assert(marker>0) : "Estrazione da Pila vuota";  
    return contenuto[--marker];  
}
```



La Pila in Java - 3

```
private void cresci(){
    Object temp []=new Object[size];
    for (int k=0;k<size;k++)
        temp [k]=contenuto [k];
    contenuto=new Object[size+DEFAULTGROWTHSIZE];
    for (int k=0;k<size;k++)
        contenuto [k]=temp [k];
    size+=DEFAULTGROWTHSIZE;
}
```



La Pila in Java - 5

```
public static void main(String args[]) {  
    int dim=10;  
    Pila s=new Pila();  
    for (int k=0;k<dim;k++) {  
        Integer o=new Integer(k);  
        s.inserisci(o);  
    }  
    for (int k=0;k<3*dim;k++) {  
        Integer i = s.estrai();  
        int w=i.intValue();  
        System.out.println(w);  
    }  
}
```



La Pila in Java - 5

```
public static void main(String args[]) {  
    int dim=10;  
    Pila s=new Pila();  
    for (int k=0;k<dim;k++) {  
        Integer o=new Integer(k);  
        s.inserisci(o);  
    }  
    for (int k=0;k<3*dim;k++) {  
        Integer i = s.estrai();  
        int w=i.intValue();  
        System.out.println(w);  
    }  
}
```

ERRORE!
Non posso
mettere un
Object in un
Integer!



La Pila in Java - 6

```
public static void main(String args[]) {  
    int dim=10;  
    Pila s=new Pila();  
    for (int k=0;k<dim;k++) {  
        Integer o=new Integer(k);  
        s.inserisci(o);  
    }  
    for (int k=0;k<3*dim;k++) {  
        Integer i = (Integer)s.estrai();  
        int w=i.intValue();  
        System.out.println(w);  
    }  
}
```



La Pila in Java - 4

```
public static void main(String args[]) {  
    int dim=10;  
    Pila s=new Pila();  
    for (int k=0;k<dim;k++) {  
        Integer o=new Integer(k);  
        s.inserisci(o);  
    }  
    for (int k=0;k<3*dim;k++)  
        System.out.println(s.estrai());  
}  
} // end of class Pila
```



Sezione: Upcast - downcast

Upcast & downcast

```
public class Test {  
    public static void main(String a[]) {  
        new Test();  
    }  
}
```

cast

```
Test() {
```

```
    A a;
```

```
    B b = new B();
```

OK: upcast implicito

```
    a=b;
```

NO: "method f2 not found
in class A" (compiler)

```
    a.f1();
```

```
    a.f2();
```

```
}
```

```
}
```

```
class A { void f1()  
{System.out.println("f1"); } }  
class B extends A { void f2()  
{System.out.println("f2"); } }  
class C extends B { void f3()  
{System.out.println("f3"); } }
```

```
public class Test {  
    public static void main(String a[]) {  
        new Test();  
    }  
}
```

cast

```
Test() {
```

```
    A a;
```

```
    B b = new B();
```

OK: upcast implicito

```
    a=b;
```

OK: downcast corretto

```
    a.f1();
```

```
    ((B)a).f2();
```

```
}
```

```
}
```

```
class A { void f1()  
{System.out.println("f1"); } }  
class B extends A { void f2()  
{System.out.println("f2"); } }  
class C extends B { void f3()  
{System.out.println("f3"); } }
```

```
10  Dati e modelli  
public class Test {  
    public static void main(String a[]) {  
        new Test();  
    }  
}
```

cast

```
Test() {
```

```
    A a;
```

```
    B b = new B();
```

OK: upcast implicito

```
    a=b;
```

```
    a.f1();
```

NO: downcast illegitimo (runtime)
java.lang.ClassCastException

```
    ((C)a).f3();
```

Class A { void f1() }

{ System.out.println("f1"); }

```
class B extends A { void f2()
```

{ System.out.println("f2"); }

```
class C extends B { void f3()
```

{ System.out.println("f3"); }