1

# Cast control

2

# Esempio

```
public class Shape {
  public String toString() {return("i'm a Shape!");
}
class Circle extends Shape {
  public String toString() {return("i'm a Circle!");}
}
class Square extends Shape {
  public String toString() {return("i'm a Square!");}
}
class Other  {
  public String toString() {return("i'm an Other!");}
}
```

3

# controllo dei cast illegali

```
  public static void main(String[] args) {
    Shape s;
    s=new Circle();
    System.out.println(s);
    Circle c=(Circle)s;
    System.out.println(c);
    Square q=(Square)s;
    System.out.println(q);
    Other o=(Other)s;
    System.out.println(o);
  }
}
```

At run time: class cast Exception

At compile time: cannot cast Shape to Other

4

# Esempio

```
class Shape {
  public: virtual void say(){
    cout<<" I'm a shape - address="<<this<<endl;}
};
Circle:Shape {
 public: virtual void say(){
    cout<<" I'm a shape - address="<<this<<endl;}
};
class Square:Shape {
 public: virtual void say(){
    cout<<" I'm a square - address="<<this<<endl;}

};
class Other  {
 public: virtual void say(){
    cout<<" I'm a other - address="<<this<<endl;}

};
```

5

# controllo dei cast illegali

```
int main() {
    Shape * s;
    s=new Circle();
    if (s!=NULL) s->say();
    Circle * c=(Circle *)s;
    if (c!=NULL) c->say();
    Square * q=(Square *)s;
    if (q!=NULL) q->say();
    Other * o=(Other *)s;
    if (o!=NULL) o->say();
    return 0;
}
```

NESSUN CONTROLLO!

```
I'm a circle- address=008731D8
I'm a circle- address=008731D8
I'm a circle- address=008731D8
I'm a circle- address=008731D8
```

6

# static cast

```
int main() {
    Shape * s;
    s=new Circle();
    if (s!=NULL) s->say();
    Circle * c=static_cast<Circle *>(s);
    if (c!=NULL) c->say();
    Square * q=static_cast<Square *>(s);
    if (q!=NULL) q->say();
    Other * o=static_cast<Other *>(s);
    if (o!=NULL) o->say();
    return 0;
}
```

At compile time: cannot cast Shape to Other

7

# static cast

```
int main() {
    Shape * s;
    s=new Circle();
    if (s!=NULL) s->say();
    Circle * c=static_cast<Circle *>(s);
    if (c!=NULL) c->say();
    Square * q=static_cast<Square *>(s);
    if (q!=NULL) q->say();
    //Other * o=static_cast<Other *>(s);
    //if (o!=NULL) o->say();
    return 0;
}
```

Se NON virtual Shape::say()

```
I'm a shape address=008731D8
I'm a circle- address=008731D8
I'm a square- address=008731D8
```

8

# static cast

```
int main() {
    Shape * s;
    s=new Circle();
    if (s!=NULL) s->say();
    Circle * c=static_cast<Circle *>(s);
    if (c!=NULL) c->say();
    Square * q=static_cast<Square *>(s);
    if (q!=NULL) q->say();
    //Other * o=static_cast<Other *>(s);
    //if (o!=NULL) o->say();
    return 0;
}
```

Se virtual Shape::say()

```
I'm a circle address=008731D8
I'm a circle- address=008731D8
I'm a circle- address=008731D8
```

9

# dynamic cast

```
int main() {
    Shape * s;
    s=new Circle();
    if (s!=NULL) s->say();
    Circle * c=dynamic_cast<Circle *>(s);
    if (c!=NULL) c->say();
    Square * q=dynamic_cast<Square *>(s);
    if (q!=NULL) q->say();
    Other * o=dynamic_cast<Other *>(s);
    if (o!=NULL) o->say();
    return 0;
}
```

At compile time: Shape is not a class with virtual functions

Se Shape NON HA metodi virtuali

10

# dynamic cast

```
int main() {
    Shape * s;
    s=new Circle();
    if (s!=NULL) s->say();
    Circle * c=dynamic_cast<Circle *>(s);
    if (c!=NULL) c->say();
    Square * q=dynamic_cast<Square *>(s);
    if (q!=NULL) q->say();
    Other * o=dynamic_cast<Other *>(s);
    if (o!=NULL) o->say();
    return 0;
}
```

```
I'm a shape
   address=008731D8
I'm a circle-
   address=008731D8
```

Se Shape HA metodi virtuali
ma say non è virtual

11

# dynamic cast

```
int main() {
    Shape * s;
    s=new Circle();
    if (s!=NULL) s->say();
    Circle * c=dynamic_cast<Circle *>(s);
    if (c!=NULL) c->say();
    Square * q=dynamic_cast<Square *>(s);
    if (q!=NULL) q->say();
    Other * o=dynamic_cast<Other *>(s);
    if (o!=NULL) o->say();
    return 0;
}
```

```
I'm a circle
   address=008731D8
I'm a circle-
   address=008731D8
```

Se Shape HA metodi virtuali
e say è virtual

12

# Static
# e
# Arrays

13

# Una Stringa minimale

```
#include <iostream.h>
#include <string.h>
class Stringa {
  int length;
  char *c;
  public:
  static int count;
  Stringa() ;
  virtual ~Stringa();
};
int Stringa::count=0;
```

inizializzazione di una variabile statica

```
Stringa::Stringa() {
    count++;
    cout<<"Stringa ()"<<endl;
    length=1;
    c=new char[length];
    c[0]=0;
}
Stringa::~Stringa() {
    count--;
    cout<<"~Stringa"<<endl;
    delete [] c;
}
```

14

# Arrays di variabili automatiche

```
#include <iostream.h>
#include <string.h>

int main()  {
    Stringa s[5];
    cout<<Stringa::count<<endl;
    return 0;
}
```

```
Stringa()
Stringa()
Stringa()
Stringa()
Stringa()
5
~Stringa()
~Stringa()
~Stringa()
~Stringa()
~Stringa()
```

15

# Arrays di variabili dinamiche

```
#include <iostream.h>
#include <string.h>

int main()  {
    Stringa *s;
    s=new Stringa[5];
    cout<<Stringa::count<<endl;
    delete s;
    cout<<Stringa::count<<endl;
    return 0;
}
```

```
Stringa()
Stringa()
Stringa()
Stringa()
Stringa()
5
~Stringa()
4
```

Problemi con il distruttore!

16

# Arrays

```
#include <iostream.h>
#include <string.h>

int main()  {
    Stringa *s;
    s=new Stringa[5];
    cout<<Stringa::count<<endl;
    delete []s;
    cout<<Stringa::count<<endl;
    return 0;
}
```

```
Stringa()
Stringa()
Stringa()
Stringa()
Stringa()
5
~Stringa()
~Stringa()
~Stringa()
~Stringa()
~Stringa()
0
```

17

# Una Stringa minimale

```
public class Stringa {
  private int length;
  private char c[];
  public static int count=0;
  public Stringa() {
    count++;
    System.out.println("Stringa ()");
    length=1;
    c=new char[length];
    c[0]=0;
  }
  public void finalize() {
    count--;
    System.out.println("Stringa ()");
  }
```

inizializzazione di una variabile statica

18

# Arrays di variabili automatiche

```
public static void main(String a[]) {
    Stringa s[]=new Stringa[5];
    System.out.println(Stringa.count);
    s=null;
    System.gc();
    System.out.println(Stringa.count);
 }
```

```
0
0
```

19

# Arrays di variabili automatiche

```
public static void main(String a[]) {
    Stringa s[]=new Stringa[5];
    System.out.println(Stringa.count);
   for (int i=0;i<5;i++)
      s[i]=new Stringa();
    s=null;
    System.gc();
    System.out.println(Stringa.count);
 }
```

creazione individuale degli elementi del vettore

```
Stringa()
Stringa()
Stringa()
Stringa()
Stringa()
5
~Stringa()
~Stringa()
~Stringa()
~Stringa()
~Stringa()
0
```