1

# Pila 1

```
public class Pila {

 protected int size;
 protected int defGsize;
 protected int marker;
 protected int contenuto[];
 final int initialSize=3;
```

```
class Pila {

 protected:
  int size;
  int defGsize;
  int marker;
  int * contenuto;
  const int initialSize=3;
```

2

# Pila 2

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

public Pila(int s) {
  size=s;
  defGsize =s;
  marker=0;
  contenuto=new int[size];
}
```

```
public:
    Pila() {
    size=initialSize;
    defGsize =initialSize;
    marker=0;
    contenuto=new int[size];
  }

  Pila(int s) {
    size=s;
    defGsize =s;
    marker=0;
    contenuto=new int[size];
  }

  ~Pila() {
    delete [](contenuto);
  }
```

3

# Pila 3

Attenti all'ordine!

```
private:
  void Pila::cresci(){
   size+=dfg;
    int * temp=new int[size];
    for (int k=0; k<=marker;k++)
       temp[k]=contenuto[k];
    delete [](contenuto);
    contenuto=temp;
}
```

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

4

# Pila 4

```
public:
  void inserisci(int k){
    if (size==marker) cresci();
    contenuto[marker]=k;
    marker++;
  }
  int estrai() {
    assert(marker>0);
    return contenuto[--marker];
  }
```

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

5

# Pila 5

main fuori dalla classe

```
}
int main() {
  Pila *p=new Pila;
  for (int k=0;k<5;k++)
    p->inserisci(k);
 for (int k=0;k<5;k++)
    cout<<p->estrai()<<endl;
}
```

```
  public static void main(String[] a) {
    Pila p=new Pila;
    for (int k=0;k<5;k++)
      p.inserisci(k);
    for (int k=0;k<5;k++)
      System.out.println(p.estrai());
  }
}
```

main dentro la classe

6

# Altre differenze

- Package
- Una classe per file

- Separazione tra interfaccia (.h) e implementazione (.cpp)
- Pila::Pila(){…} ecc.
- Preprocessore:
  - #include
  - #define #ifdef

7

# Catch up...

Ereditarietà

Concatenamento dei costruttori

Polimorfismo – upcast/downcast

Gerarchia standars (Object…)

Modificatori

Arrays di oggetti

Type fuctionality (equals, clone, toString)

8

# Ereditarietà

```
public class Coda extends Pila{
  public int estrai() {
     assert(marker>0):"Coda vuota";
     int retval=contenuto[0];
     for (int k=1;k<marker;k++;)
       contenuto[k-1]=contenuto[k];
     marker--;
     return retval;
    }
}
```

```
class Coda:public Pila{
  public:
    int estrai();
}
int Coda::estrai() {
     assert(marker>0);
     int retval=contenuto[0];
     for (int k=1;k<marker;k++;)
       contenuto[k-1]=contenuto[k];
     marker--;
     return retval;
    }
}
```

9

# Ereditarietà

```
int main() {
   Pila * p=new Pila;
   for (int k=0;k<6;k++) {
        p->inserisci(k);
   }
   for (int k=0;k<6;k++)
        cout<<p->estrai()<<" ";
   delete p;
   cout<<endl;
   Coda * c=new Coda();
   for (int k=0;k<6;k++) {
        c-> inserisci(k);
   }
   for (int k=0;k<6;k++)
        cout<<c->estrai ()<<" ";
   delete c;
    return 0;
}
```

```
5 4 3 2 1 0
0 1 2 3 4 5
```

10

# Ereditarietà

```
int main() {
   Pila * p=new Pila;
   for (int k=0;k<6;k++) {
        p->inserisci(k);
   }
   for (int k=0;k<6;k++)
        cout<<p->estrai()+" ";
   delete p;
   cout<<endl;
   Coda * c=new Coda();
   for (int k=0;k<6;k++) {
        c-> inserisci(k);
   }
   for (int k=0;k<6;k++)
        cout<<c->estrai ()+" ";
   delete c;
    return 0;
}
```

```
5 4 3 2 1 0
0 1 2 3 4 5
```

11

# Costruttori

In Java il costruttore della sottoclasse invoca il costruttore void della superclasse, a meno si usare super.

Che succede in C++?

```
Pila::Pila() {
     cout<<"Pila()"<<endl;
     size=initialSize;
     defGsize =initialSize;
     marker=0;
     contenuto=new int[size];
}
Pila::Pila(int s) {
     cout<<"Pila(int)"<<endl;
     size=s;
     defGsize =s;
     marker=0;
     contenuto=new int[size];
}
Coda::Coda() {
     cout<<"Coda()"<<endl;
}
Coda::Coda(int s) {
     cout<<"Coda(int)"<<endl;
}
```

12

# Costruttori

```
Pila()
Coda()
```

```
int main() {
  Coda *c=new Coda;
  return 0;
}
```

```
Pila()
Coda(int)
```

```
int main() {
  Coda *c=new Coda(3);
  return 0;
}
```

13

# Costruttori

Costrutto equivalente al super di Java

```
Coda::Coda() {
     cout<<"Coda()"<<endl;
}
Coda::Coda(int s):Pila(s) {
     cout<<"Coda(int)"<<endl;
}
```

```
Pila()
Coda()
```

```
int main() {
  Coda *c=new Coda();
  return 0;
}
```

```
Pila(int)
Coda(int)
```

```
int main() {
  Coda *c=new Coda(3);
  return 0;
}
```

14

# Distruttori – finalize()

```
public class Pila{
...
  public void finalize() {
    System.out.println("P");
  }
}
public class Coda extends Pila{
...
  public void finalize() {
    System.out.println("C");
  }
}
...
public static void main(String []a){
  Coda c=new Coda();
  c=null;
  System.gc();
}
```

```
C
```

15

# Distruttori – finalize()

```
public class Pila{
...
  public void finalize() {
    System.out.println("P");
  }
}
public class Coda extends Pila{
...
  public void finalize() {
    System.out.println("C");
    super.finalize();
  }
}
...
public static void main(String []a){
  Coda c=new Coda();
  c=null;
  System.gc();
}
```

```
C
P
```

16

# Distruttori

In Java l'implementazione di finalize() è rara, e la sua chiamata avviene automaticamente.

In Java il finalize della sottoclasse NON chiama il finalize della superclasse, a meno di usare super.

```
Pila::~Pila() {
     cout<<"~Pila"<<endl;
   delete [] contenuto;
}
Coda::~Coda() {
     cout<<"~Coda()"<<endl;
}
```

Che succede in C++?

17

# Distruttori

```
Pila()
~Pila()
```

```
int main() {
  Pila *c=new Pila;
  delete c;
  return 0;
}
```

```
Pila()
Coda()
~Coda()
~Pila()
```

```
int main() {
  Coda *c=new Coda;
  delete c;
  return 0;
}
```

```
Pila()
Coda()
~Pila()
```

```
int main() {
  Pila *c=new Coda;
  delete c;
  return 0;
}
```

18

# Distruttori virtuali

```
Class Pila{
public:
…
   virtual Pila::~Pila() ;
…
}
Pila::~Pila() {
    cout<<"~Pila"<<endl;
   delete [] contenuto;
}
Coda::~Coda() {
    cout<<"~Coda()"<<endl;
}
```

19

# Distruttori virtuali

```
Pila()
Coda()
~Coda()
~Pila()
```

```
int main() {
  Coda *c=new Coda;
  return 0;
}
```

Dichiarate SEMPRE virtual il distruttore…

20

# Polimorfismo

```
Pila()
Coda()
5 4 3 2 1 0
~Coda()
~Pila()
```

```
int main() {
   Pila * p;
   p=new Coda();
   for (int k=0;k<6;k++) {
        p-> inserisci(k);
   }
   for (int k=0;k<6;k++)
        cout<<p->estrai()<<endl;
   delete p;
   return 0;
}
```

21

# Polimorfismo- virtual

```
Pila()
Coda()
0 1 2 3 4 5
~Coda()
~Pila()
```

```
class Pila{
  public:
    virtual int estrai();
}

int main() {
   Pila * p;
   p=new Coda();
   for (int k=0;k<6;k++) {
        p->inserisci(k);
   }
   for (int k=0;k<6;k++)
        cout<<p->estrai()<<endl;
   delete p;
   return 0;
}
```