

# Composition and initialization

- Version 1: Dr. Ofir Pele
- Version 2: Dr. Miri Ben-Nissan
- Version 3: Dr. Erel Segal-Halevi

## Composition

```
class Line {  
    Point p1, p2;  
};
```

C++	Java
Members are <b>objects</b>	Members are <b>pointers</b>

C++

Java

Stack:

line.p1.x

line.p1.y

line.p2.x

line.p2.y

Heap:

line

p1

p2

p1.x

p1.y

p2.x

p2.y

```
int main () { Line line; }
```

# Construction

```
int main() {  
    B b;  
}
```

```
class A {  
public:  
    A() {  
        std::cout <<  
            "A - " <<  
            "parameterless" <<  
            " ctor\n";  
    }  
};
```

```
class B {  
    A _a1, _a2;  
public:  
    B() {  
        std::cout <<  
            "B - parameterless " <<  
            "ctor\n";  
    }  
};
```

**Question: In what order  
are the constructors  
called?**

# Construction order

```
int main() {  
    B* pb;  
}
```

```
class A {  
public:  
    A() {  
        std::cout <<  
            "A - " <<  
            "parameterless" <<  
            " ctor\n";  
    }  
};
```

```
class B {  
    A _a1, _a2;  
public:  
    B() {  
        std::cout <<  
            "B - parameterless " <<  
            "ctor\n";  
    }  
};
```

```
// Answer: small to large  
A - parameterless ctor  
A - parameterless ctor  
B - parameterless ctor
```

# Destruction

```
int main() {  
    B b;  
}
```

```
class A {  
public:  
    ~A() {  
        std::cout <<  
            "A - " <<  
            " dtor\n";  
    }  
};
```

```
class B {  
    A _a1, _a2;  
public:  
    ~B() {  
        std::cout <<  
            "B - dtor\n";  
    }  
};
```

Question: In what order  
are the destructors  
called?

# Destruction order

```
int main() {  
    B b;  
}
```

```
class A {  
public:  
    A() { ... }  
    ~A() {  
        std::cout <<  
            "A - " <<  
            " dtor\n";  
    }  
};
```

```
class B {  
    A _a1, _a2;  
public:  
    B() { ... }  
    ~B() {  
        std::cout <<  
            "B - dtor\n";  
    }  
};
```

```
// Answer: large to small  
B - dtor  
A - dtor  
A - dtor
```

# The parameterless ctor (aka default ctor)

```
int main() {  
    B b;  
}
```

```
class A {  
public:  
    A(int a) {  
        std::cout <<  
            "A ctor with one  
            parameter\n";  
    }  
};
```

```
class B {  
    A _a1, _a2;  
public:  
    B() {  
        std::cout <<  
            "B - parameterless " <<  
            "ctor\n";  
    }  
};
```

```
// compilation error  
No parameterless ctor for _a1, _a2
```



# The initialization list

```
int main() {  
    B b(2);  
}
```

```
class A {  
public:  
    A(int a) {  
        std::cout <<  
            "A (" << a << ") "  
        << std::endl;  
    }  
};
```

```
class B {  
    A _a1, _a2;  
public:  
    B(int i)  
        : _a1 (i), _a2(2*i)  
    {  
        std::cout  
        << "B cons"  
        << std::endl;  
    }  
};
```

```
// output  
A (2)  
A (4)  
B cons
```

# Initialization using pointers - Java-like

```
int main() {  
    B b(2);  
}
```

```
class A {  
public:  
    A(int a) {  
        std::cout <<  
            "A (" << a << ") "  
        << std::endl;  
    }  
};
```

```
class B {  
    A *_ap;  
public:  
    B(int i);  
};  
  
B::B(int i) {  
    _ap = new A (i);  
    cout << "B cons\n";  
}
```

```
// output  
A (2)  
B cons
```

# Initialization using pointers (2)

```
int main() {  
    B b(2);  
}
```

```
class A {  
public:  
    A(int a) {  
        std::cout <<  
            "A (" << a << ") "  
        << std::endl;  
    }  
};
```

```
class B {  
    A *_ap;  
public:  
    B(int i);  
    ~B();  
};  
  
B::B(int i)  
    : _ap (new A(i))  
{  
    cout << "B cons\n";  
}
```

```
output //  
A (2)  
B cons
```

# Advantages of using initialization list *(folder 0)*

1. Faster – no need for default initialization.
2. Safer – be sure that all components are ready.
3. Can initialize constants and reference variables.
4. Can initialize parent class.