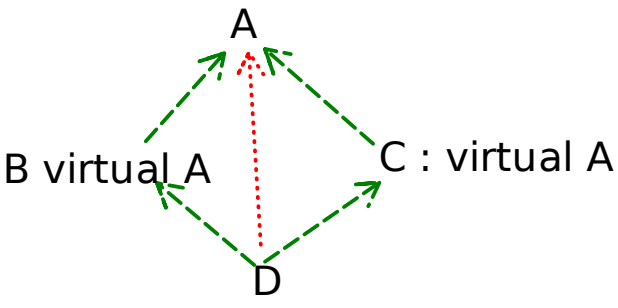
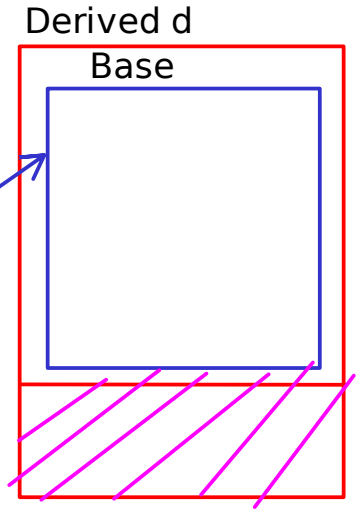
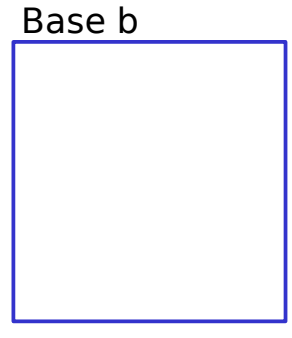


A -> members-> multiple times inherited -> D

virtual A -> members -> members inherited only once -> D

```
class Base{

}

class Derived : public Base{

}
```






```
Base *bptr = new Derived(); // upcasting
```

bptr-> access only base class members

```
Derived *dptr = (Derived *)bptr; // Downcasting
```

dptr-> access both members of base + derived

```
Base{
void f1(){}
}

Derived {
// function overriding
void f1(){}
}
```

```
bptr->f1(); // Base -> Early binding

dptr->f1(); // Derived -> Early binding
```

Compiler decides the function call at compile time looking at the type of pointer

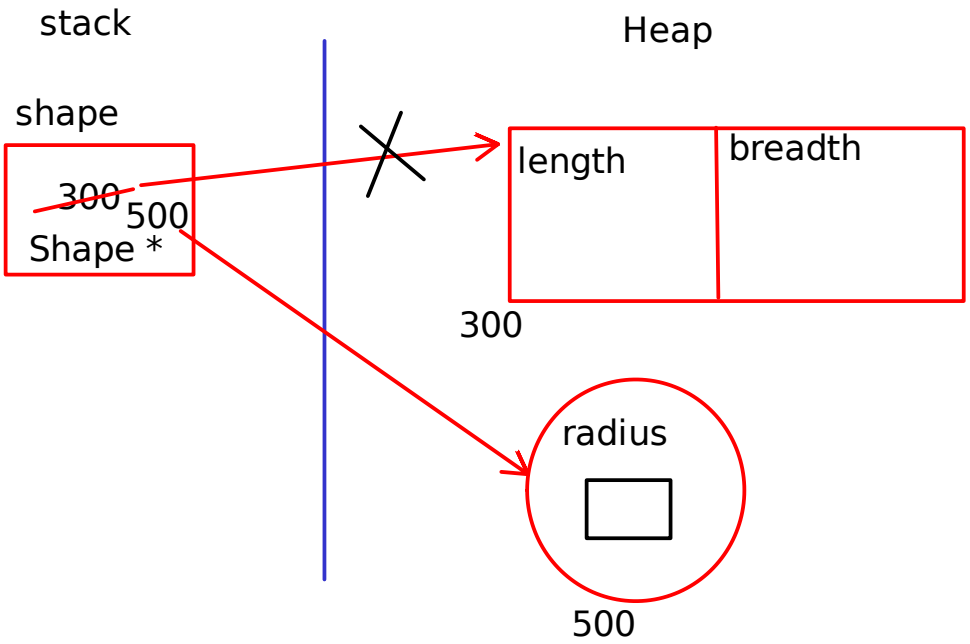
```
Base{
virtual void f1(){}
}

Derived {
// function overriding
void f1(){}
}
```

```
Base *bptr = new Base();
bptr-> f1(); // Base

bptr = new Derived();
bptr->f1(); // Derived - Late Binding
```

Compiler decides the function call at run time time looking at the type of object

```
// abstract class
Shape{
// pure virtual function
virtual void calculateArea()=0;
}

Circle : public Shape{
// function overriding
void calculateArea(){
}
}

Rectangle : public Shape{
// function overriding
void calculateArea(){
}
}
```

```
Shape shape; //NO cannot create object of abstract class

Shape *shape;//yes can create pointer of abstract class


 Circle circle; // Yes

 Rectangle rect;// Yes


Shape *shape = NULL;
// shape = new Rectangle(); // Upcasting
// shape = new Circle(); // Upcasting

shape-> calculateArea();

delete shape;
```




```
shape = new Circle(); //shape = 500;
shape->calculateArea();
delete shape; // delete 500;

Circle *c = (Circle *)shape
```

delete 500

```
// abstract class
Shape{
// pure virtual function
virtual void calculateArea()=0;
}

Circle : public Shape{
// function overriding
void calculateArea(){
}
}

Rectangle : public Shape{
// function overriding
void calculateArea(){
}
void calculatePerimeter(){
}
}
```

```
Shape *shape;
shape = new Rectangle();
//shape = new Circle();
    shape->calculateArea();

    if(obj is of rectangle){
    Rectangle *rect = (Rectangle *)shape;
    rect->calculatePerimeter();
    }
```

dynamic_cast
static_cast
reinterpret_cast
const_cast

dynamic_cast -> type_id(*shape)-> polymorphic

static_cast -> bit risker -> dev should be 100% sure about upcasting

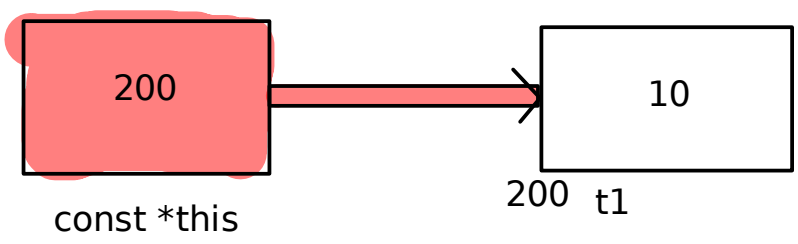
reinterpret_cast -> very risker -> used to convert one type of pointer into another type

```
Test t1;
t1.f2();
//Test *const this = &t1;
const Test *const this = &t1;
```

```
Test t2;
this = &t2;
```

type qualifiers
const, volatile

type modifiers
long short

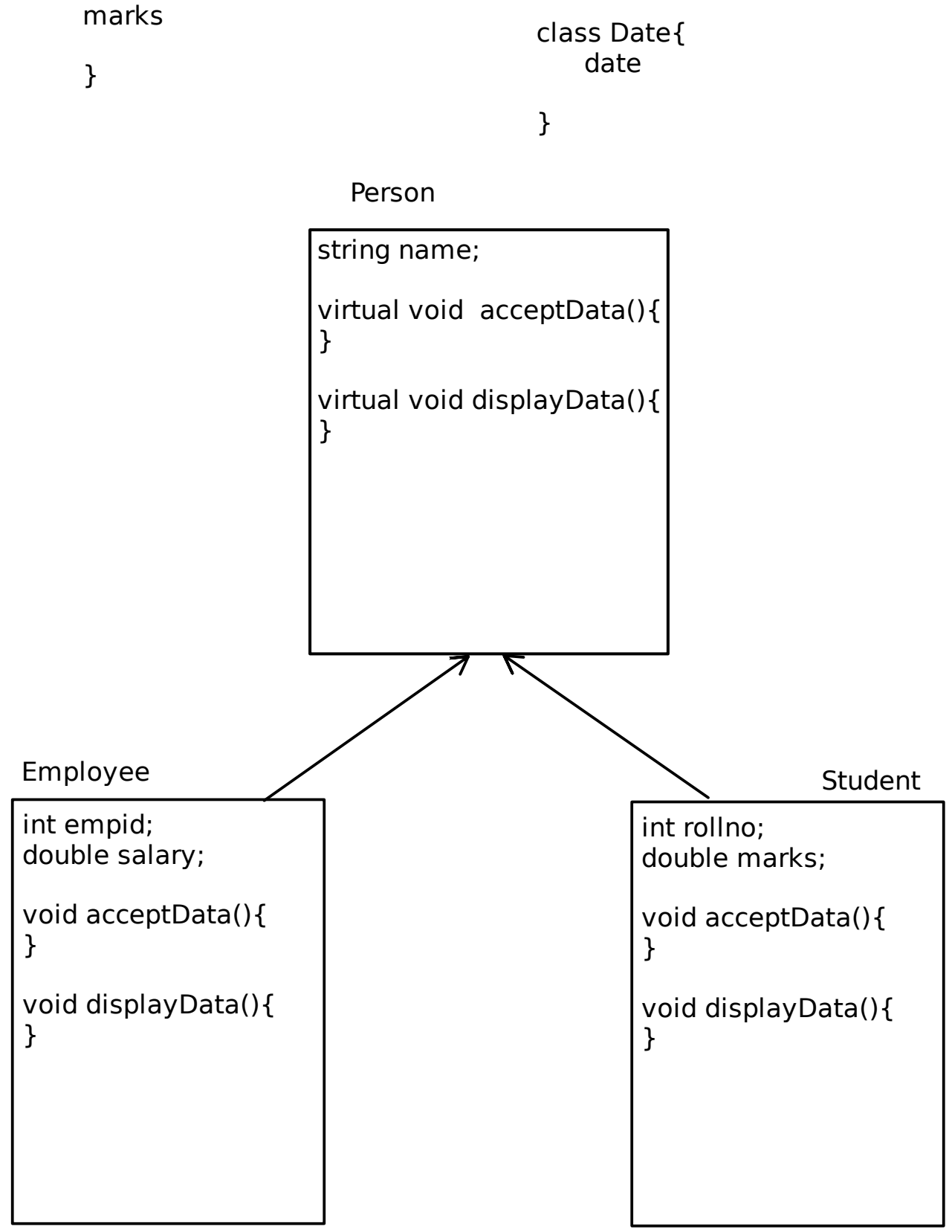


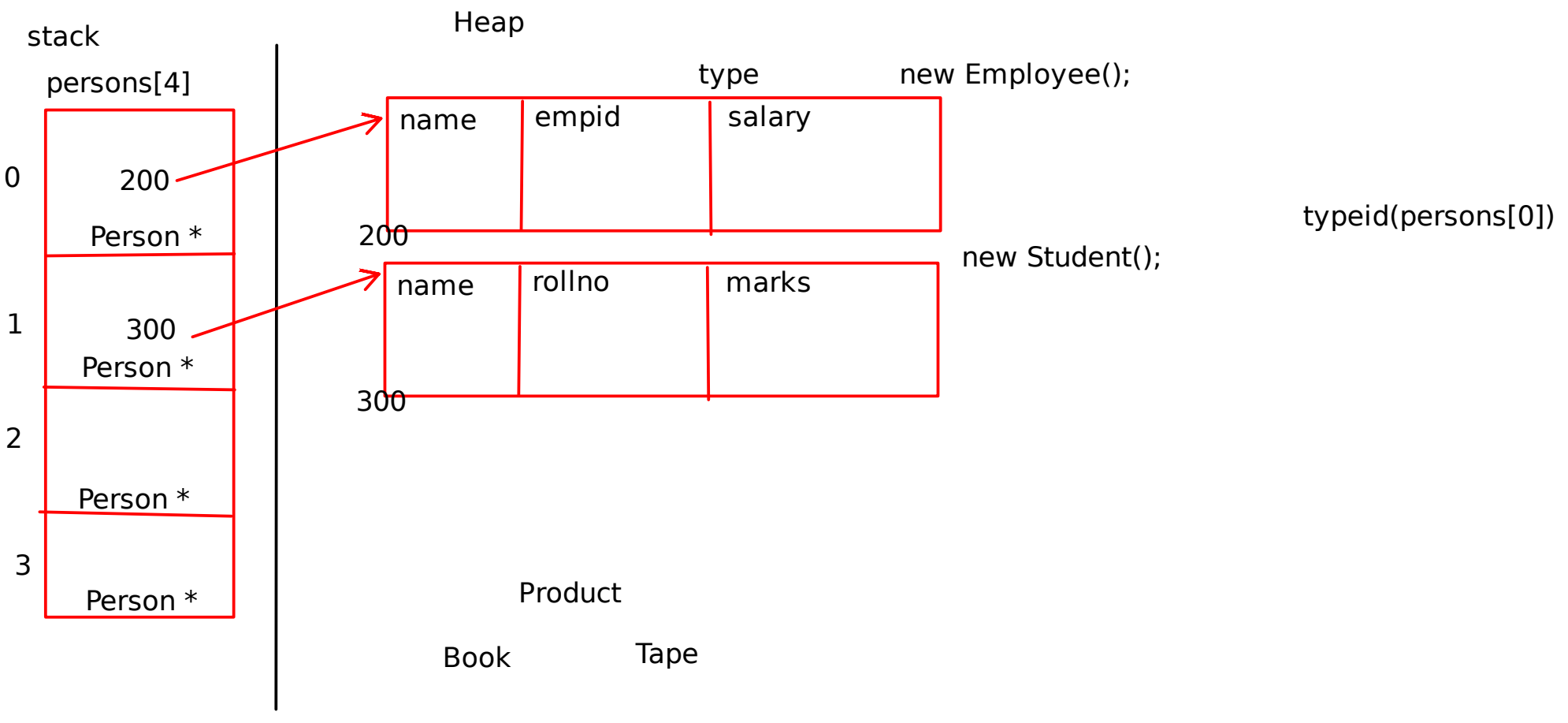
//*this = 20;// NOT OK

```
class Person{
 name
date
}
        class Date{
            date

        }
```

```
Employee:Person{
empid
salary

}
```

```
Student:Person{
rollno
marks

}
```





```
Product{
id,
title,
price
}

Book:Product{
author
}

Product arr*[3];
```

```
virtual void accept(){
}

virtual void display(){
}

double getPrice(){
return price;
}

Tape:Product{
artist
}
```

```
calculateFinalBill(){
total_bill = 0;
tape_total = 0;
books_total = 0;

for(int i=0;i<3;i++){
if(typeid(*arr[i])== typeid(Book))
      books_total= books_total + arr[i]->getPrice();

if(typeid(*arr[i])==typeid(Tape))
      tape_total = tape_total + arr[i]->getPrice();


total bill = (books_total * 0.9) + ()...

}

}
```