



BITS Pilani
Pilani Campus

Object Oriented Programming

CS F213

Dr. Amitesh Singh Rajput
Dr. Amit Dua

Questions from previous class



1. Can we call the constructor of the parent class directly?
2. Can we call the constructor of grand parent class directly?
3. Can we call any constructor of any class directly?

No, you cannot **call** a **constructor** from a **method**.

The only place from which you **can** invoke **constructors** using “this()” or “super()” **is** the first line of another **constructor**. If you try to invoke **constructors** explicitly elsewhere, a compile time error **will** be generated.

®:tutorialspoint

Questions



How can we use `super` to call the grandparent class? Can we use `super.super.method()`?

Multi Level Inheritance



```
class Grandparent {  
    public void Print() {  
        System.out.println("Grandparent's Print()");  
    }  
}
```

```
class Parent extends Grandparent {  
    public void Print() {  
        System.out.println("Parent's Print()");  
    }  
}
```

```
class Child extends Parent {  
    public void Print() {  
        super.super.Print();  
        System.out.println("Child's Print()");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Child c = new Child();  
        c.Print();  
    }  
}
```

Multi Level Inheritance



```
class Grandparent {  
    public void Print() {  
        System.out.println("Grandparent's Print()");  
    }  
}
```

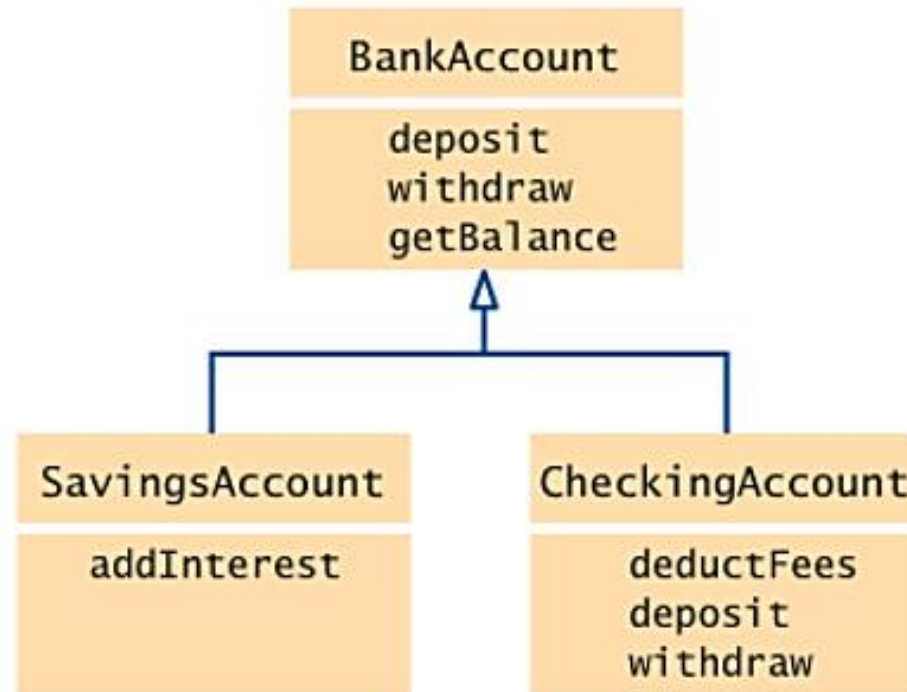
```
class Parent extends Grandparent {  
    public void Print() {  
        super.Print();  
        System.out.println("Parent's Print()");  
    }  
}
```

```
class Child extends Parent {  
    public void Print() {  
        super.Print();  
        System.out.println("Child's Print()");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Child c = new Child();  
        c.Print();  
    }  
}
```

Grandparent's Print() Parent's Print() Child's Print()
--

Bank Inheritance Scenario



Single Inheritance - Example



```
class BankAccount{  
    private int acc;  
    private String name;  
    private float amount;
```

```
BankAccount(int acc,String name,float amt){  
    this.acc = acc;  
    this.name = name;  
    this.amount = amt;  
}
```

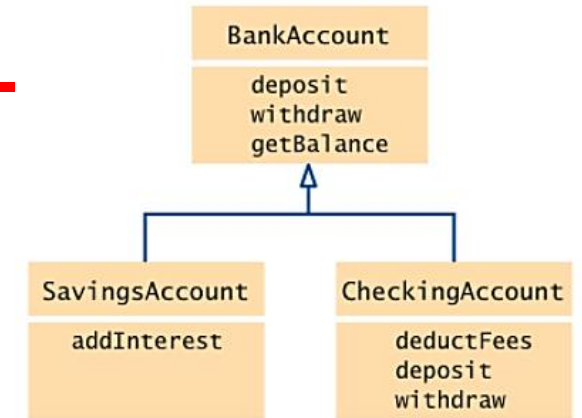
```
void setAcc(int acc){  
    this.acc = acc;  
}
```

```
void setName(String name){  
    this.name = name;  
}
```

```
float getBalance(){  
    return amount;  
}
```

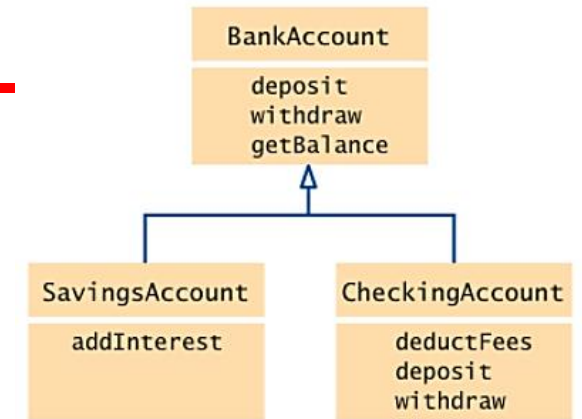
```
void deposit(float amount){  
    this.amount = this.amount + amount;  
}
```

```
void withdraw(float amount){  
    if (this.amount < amount)  
        System.out.println("Insufficient Funds");  
    else  
        this.amount = this.amount - amount;  
}
```



Single Inheritance - Example

```
class SavingsAccount extends BankAccount{  
    private float interest;  
  
    SavingsAccount(int acc,String name,float amt,float interest){  
        super(acc,name,amt);  
        this.interest = interest;  
    }  
  
    void addInterest(){  
        float interest = getBalance()*this.interest /100;  
        deposit(interest);  
    }  
}
```



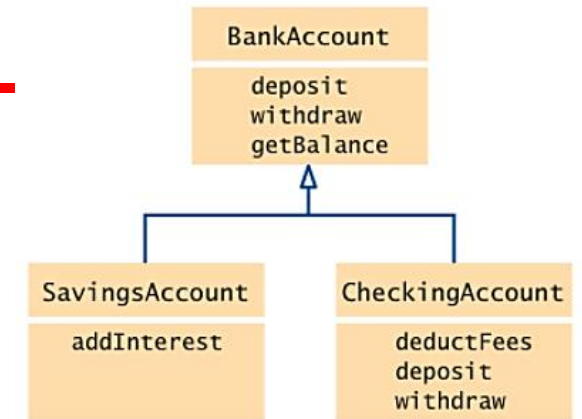
Single Inheritance - Example

```
class TestAccount{
    public static void main(String[] args) {
        SavingsAccount sa = new SavingsAccount(111,"Ankit",5000,9);
        System.out.println("Initial: "+sa.getBalance());

        sa.deposit(1000);
        System.out.println("After Deposit: " + sa.getBalance());

        sa.addInterest();
        System.out.println("Deposit+Interest: " + sa.getBalance());

        sa.withdraw(6000);
        System.out.println("After Withdraw: " + sa.getBalance());
    }
}
```



Initial: 5000.0
After Deposit: 6000.0
Deposit + Interest: 6540.0
After Withdraw: 540.0



BITS Pilani
Pilani Campus

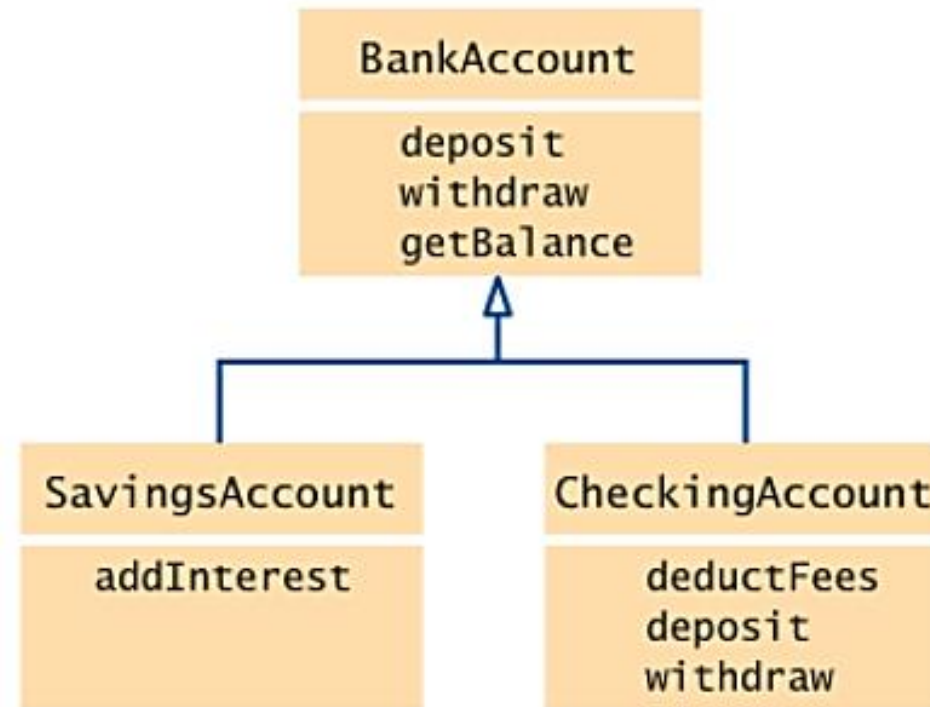
Method Overriding

What is Overriding?



- In a class hierarchy, when **a method in a subclass has the same name and type signature as a method in its superclass**, then the method in the subclass is said to override the method in the superclass.
- When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass.
- The version of the method defined by the superclass will be hidden.
- A subclass may call an overridden superclass method by prefixing its name with the 'super' keyword and a dot (.).

Bank Inheritance Scenario



Overriding - Example



```
class CheckingAccount extends BankAccount{
    private static final float TRANS_FEE = 25;
    private static final int FREE_TRANS = 2;
    private float TransCount = 0;

    CheckingAccount(int acc,String name,float amt) {
        super(acc,name,amt);
    }

    void deductFee()    {
        if(TransCount > FREE_TRANS){
            float fee = (TransCount-FREE_TRANS)*TRANS_FEE;
            withdraw(fee);
            TransCount = 0;
        }
    }
}
```

```
void deposit(float amount){
    TransCount++;
    super.deposit(amount);
}

void withdraw(float amount){
    TransCount++;
    super.withdraw(amount);
}
}
```

Overriding - Example



```
class TestAccount{  
    public static void main(String[] args) {  
        CheckingAccount ca = new CheckingAccount(111,"Ankit",5000);  
        System.out.println("Initial: "+ca.getBalance());  
        ca.deposit(1000);  
        ca.withdraw(2000);  
        ca.deposit(6000);  
        System.out.println("After three Transactions: " + ca.getBalance());  
        ca.deductFee();  
        System.out.println("After fee Deduction: " + ca.getBalance());  
    }  
}
```

Initial: 5000.0
After three Transactions: 10000.0
After fee Deduction: 9975.0



BITS Pilani
Pilani Campus

‘Final’ Keyword

Java Final Keyword



- Makes variable a constant
- Prevents Method Overriding
- Prevents Inheritance

Blank or uninitialized final variable



- A final variable in Java can be assigned a value only once.
- We can assign a value either in declaration or later.
- A blank final variable in Java is a final variable that is **not initialized during declaration**.
- It can be initialized only once (preferably within a constructor).

Final blank variable



Example 1:

```
class first{
    public static void main(String args[]){
        final int i;
        i = 10;
        System.out.println("s1: "+i);
        i = 20; // Error
    }
}
```

Example 2:

```
class first{
    final int i;
    i = 10 // Error
    first(){
        i = 10;
    }
    public static void main(String args[]){
        System.out.println("s1:"+ new first().i);
    }
}
```

Static Blank Final Variable



- A static final variable that is not initialized at the time of declaration is known as static blank final variable.
- It can be initialized only in static block.

```
class A{  
    static final int data;      //static blank final variable  
    static{  
        data = 50;  
    }  
    public static void main(String args[]){  
        System.out.println(A.data);  
    }  
}
```

Questions?



- Can we inherit final method?
 - YES. But it cannot be overridden.
- Can we declare a constructor final?
 - NO. Constructor is not inherited

Questions



- Does a subclass object creation always lead to parent class object creation?
- Does constructor execution always lead to object creation?
- If `super()` is not defined in child class constructor, and parent class has overloaded constructor, which constructor of parent is called?
- If `super()` is not defined in child class, and parent class also does not have a constructor with no arguments, what happens?
- How can we access an overridden method() from its grand parent class.
- What is by run time polymorphism and how is it different from compile time polymorphism?



BITS Pilani
Pilani Campus

Run Time Polymorphism

Dynamic Method Dispatch



- **Method overriding** is one of the ways in which Java supports Runtime Polymorphism.
- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
- An overridden method is called through the reference variable of a superclass.
- The determination of the method to be called is based on the **object being referred to by the reference variable**.
- **Upcasting**: The reference variable of Parent class refers to the object of Child class.

Example



```
public class Parent {  
    void show(){  
        System.out.println("In parent class");  
    }  
}
```

```
public class Child extends Parent{  
    void show(){  
        System.out.println("In child class");  
    }  
    public static void main(String args[]){  
        Parent obj = new Child();  
        obj.show();  
    }  
}
```

Output:

In child class

Static vs. Dynamic Binding (Early vs. Late Binding)



- Static binding happens at compile-time while dynamic binding happens at runtime.
- Binding of private, static and final methods always happen at compile time since these methods cannot be overridden.
- When the method overriding is actually happening and the reference of parent type is assigned to the object of child class type then such binding is resolved during runtime.
- The binding of overloaded methods is static and the binding of overridden methods is dynamic.



BITS Pilani
Pilani Campus

Thank You!