

Praktikum inheritance

Nama : Tegar Pratama

NIM : 20210040036

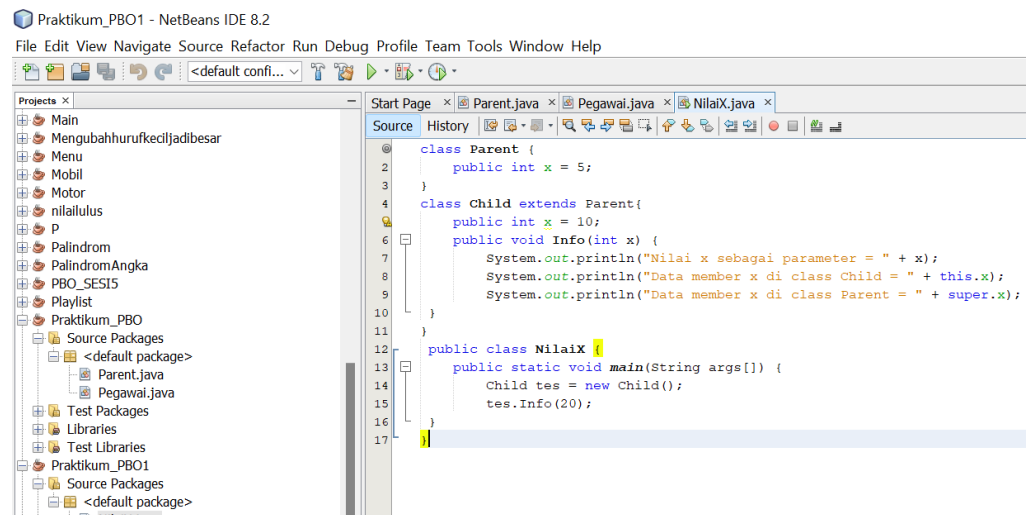
Kelas : TI21A

1. Kode Program upload ke github dengan nama repository praktikum-inheritance
2. Berikan analisa setiap percobaan dalam bentuk File teks pdf dan upload juga ke github praktikum-inheritance

JAWAB:

Percobaan berikut ini menunjukkan penggunaan kata kunci “super”.

- PERCOBAAN 1 INPUT

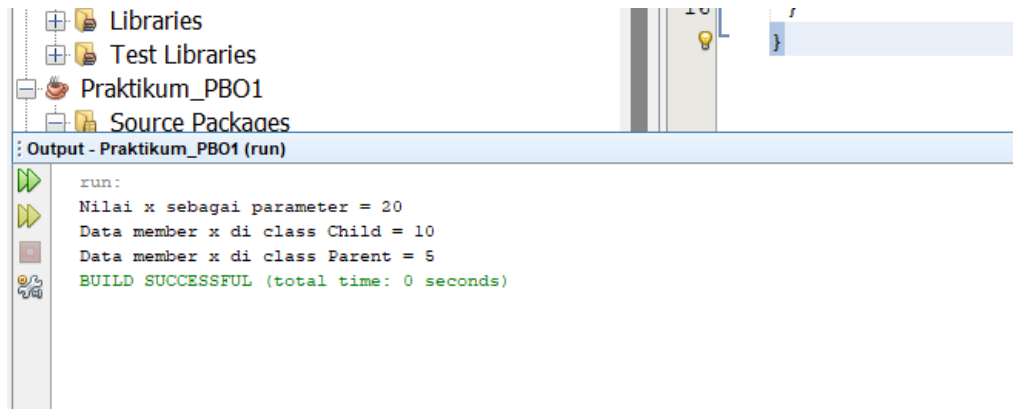


```
class Parent {
    public int x = 5;
}

class Child extends Parent{
    public int x = 10;
    public void Info(int x) {
        System.out.println("Nilai x sebagai parameter = " + x);
        System.out.println("Data member x di class Child = " + this.x);
        System.out.println("Data member x di class Parent = " + super.x);
    }
}

public class NilaiX {
    public static void main(String args[]) {
        Child tes = new Child();
        tes.Info(20);
    }
}
```

OUTPUT



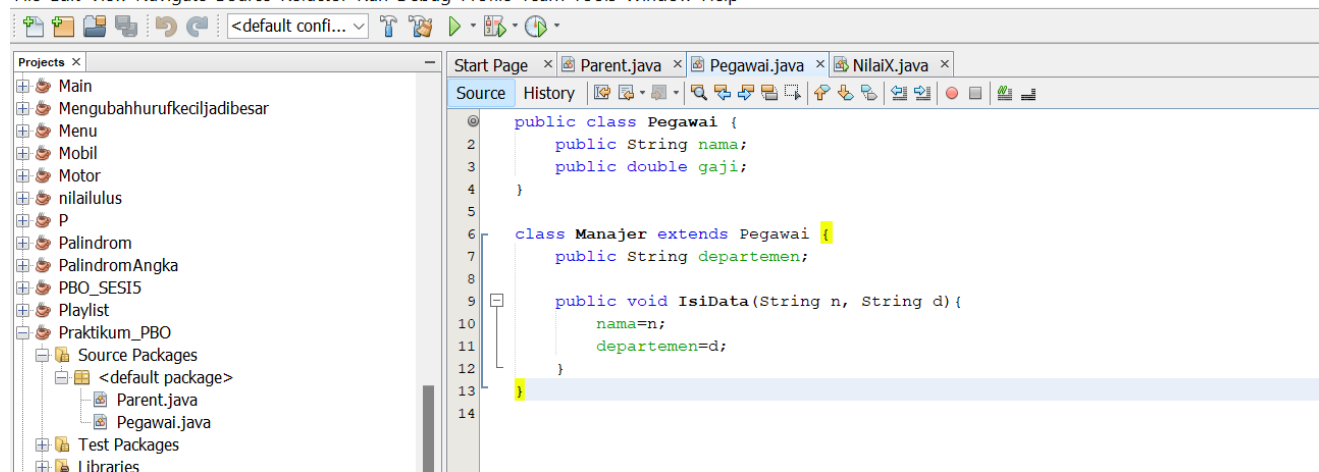
Pada percobaan satu ini Class Parent sebagai Class utama yang Mempunyai Atribut Integer bernilai 5/ int = 5, Child dikatakan sebagai sub Class Dan di dalam Class Child terdapat Parameter 20, Dimana semuanya itu ditentukan dari tes.info. dan dalam data member dari Class parent bernilai 5 Karena “Super” mengambil nilai integer dari Class parent

- PERCOBAAN 2

penggunaan kontrol akses terhadap atribut parent class. Mengapa terjadi error, dan bagaimana solusinya?

Praktikum_PBO - NetBeans IDE 8.2

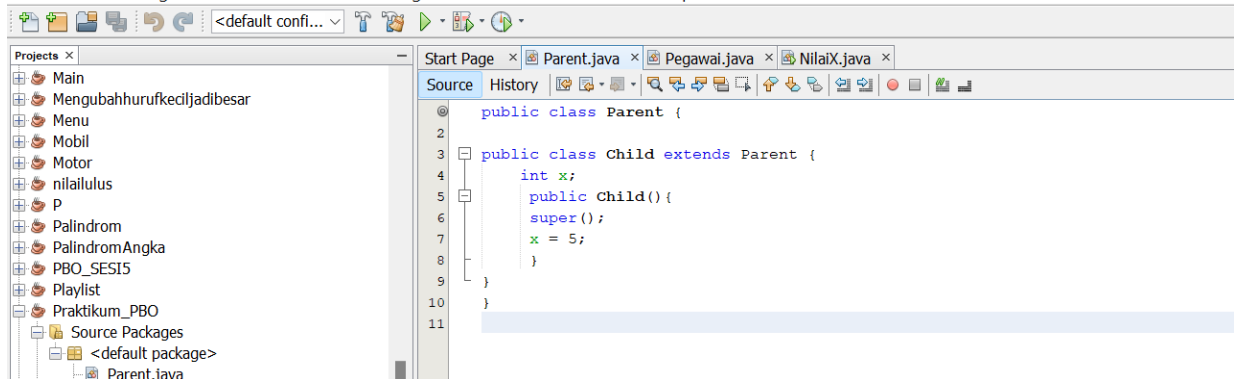
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help



Dalam penggunaan kontrol akses terhadap atribut parent class. Solusi yang saya lakukan adalah Mengubah Private String nama Menjadi Public String nama, kemudian Menghapus public di class manajer extends pegawai agar si void isidata tidak error dan kontrol akses terhadap atribu parent class tidak mengalami error

- PERCOBAAN 3

Percobaan berikut ini menunjukkan penggunaan konstruktor yang tidak diwariskan. Mengapa terjadi error, dan bagaimana solusinya?

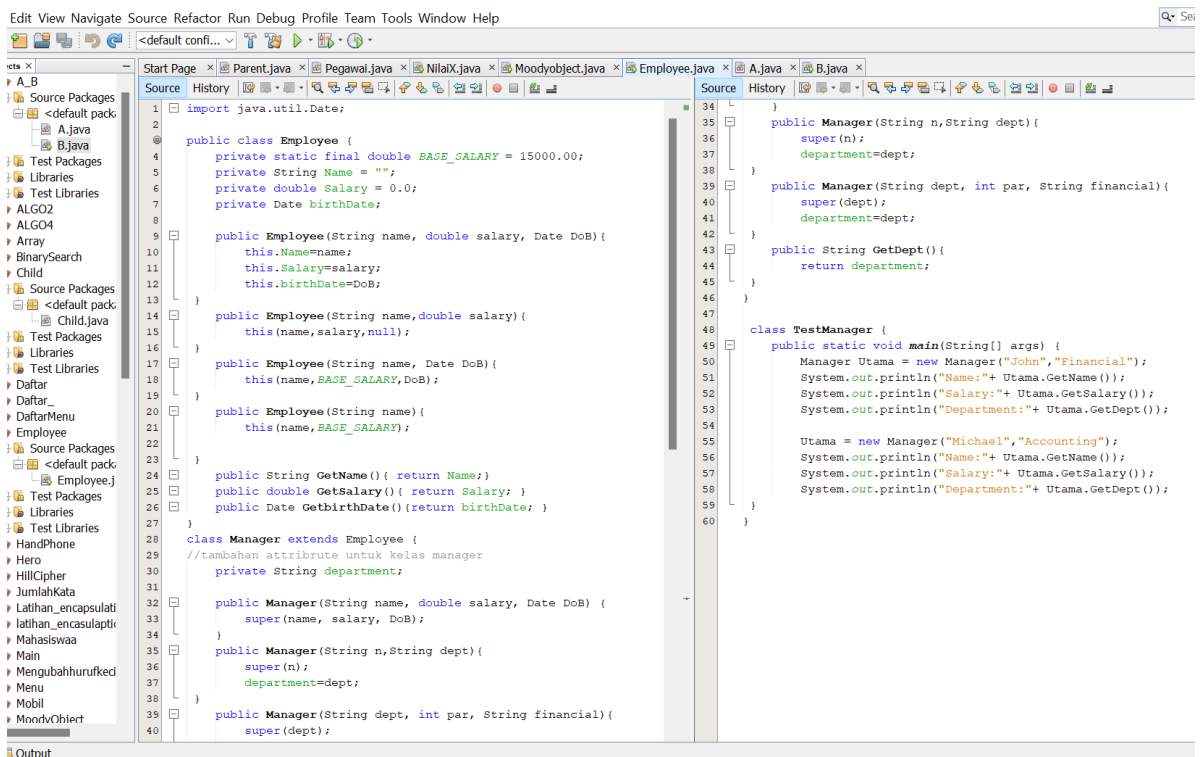


Sebelum subclass menjalankan konstruktornya sendiri, subclass akan menjalankan constructor superclass terlebih dahulu. Hal ini terjadi karena secara implisit pada constructor subclass ditambahkan pemanggilan `super()` yang bertujuan memanggil constructor superclass oleh kompiler.

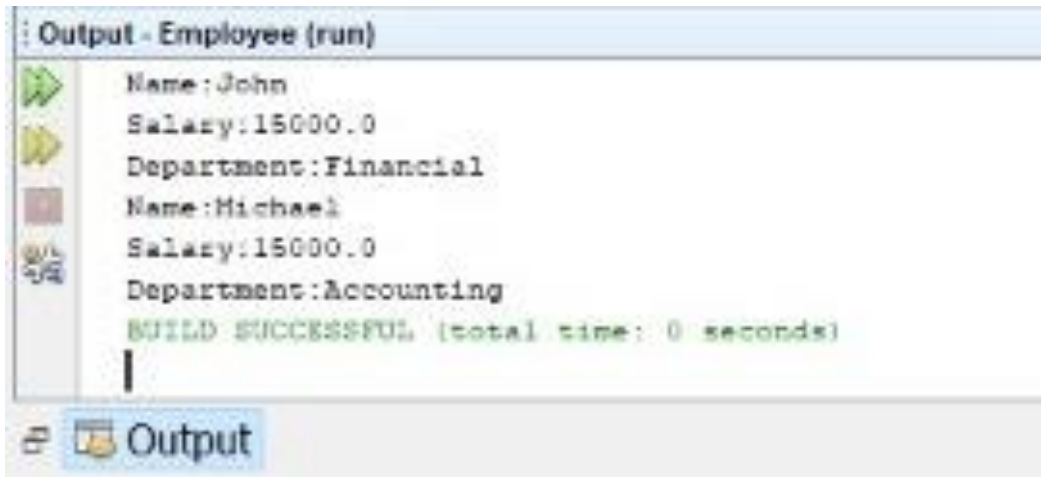
- PERCOBAAN 4

Percobaan berikut ini menunjukkan penggunaan kelas `Employee` dan subkelas `Manager` yang merupakan turunannya. Kelas `TestManager` digunakan untuk menguji kelas `Manager`.

INPUT



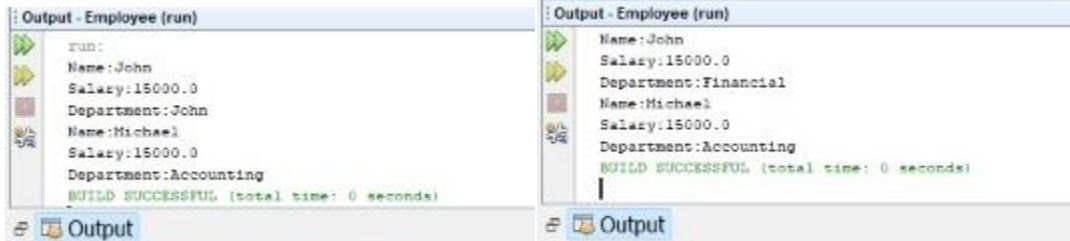
OUTPUT



```
Output - Employee (run)
Name: John
Salary: 15000.0
Department: Financial
Name: Michael
Salary: 15000.0
Department: Accounting
BUILD SUCCESSFUL (total time: 0 seconds)
```

Dari percobaan diatas public class testmanager tidak perlu memakai public lagi karena tipe dari class testmanager sudah tipe public. Untuk menunjukan output nama, gaji, dan departemen dimana sebelum departemen john hasilnya john yaitu salah, maka 500000 di dalam kodingan nya tersebut di hapus sehingga departemen john menjadi benar dan ini menandakan bahwa kelas testmanager yang digunakan untuk menguji kelas manager berjalan dengan baik.

Ini kodingan output sebelum 500000 dihapus dan sesudah



```
Output - Employee (run)
RUN:
Name: John
Salary: 15000.0
Department: John
Name: Michael
Salary: 15000.0
Department: Accounting
BUILD SUCCESSFUL (total time: 0 seconds)

Output - Employee (run)
Name: John
Salary: 15000.0
Department: Financial
Name: Michael
Salary: 15000.0
Department: Accounting
BUILD SUCCESSFUL (total time: 0 seconds)
```

- PERCOBAAN 5

Percobaan berikut ini menunjukkan penggunaan kelas MoodyObject dengan subkelas HappyObject dan SadObject. Kelas MoodyTest digunakan untuk menguji kelas dan subkelas.

- SadObject berisi :

- o sad, method untuk menampilkan pesan, tipe public

- HappyObject berisi :

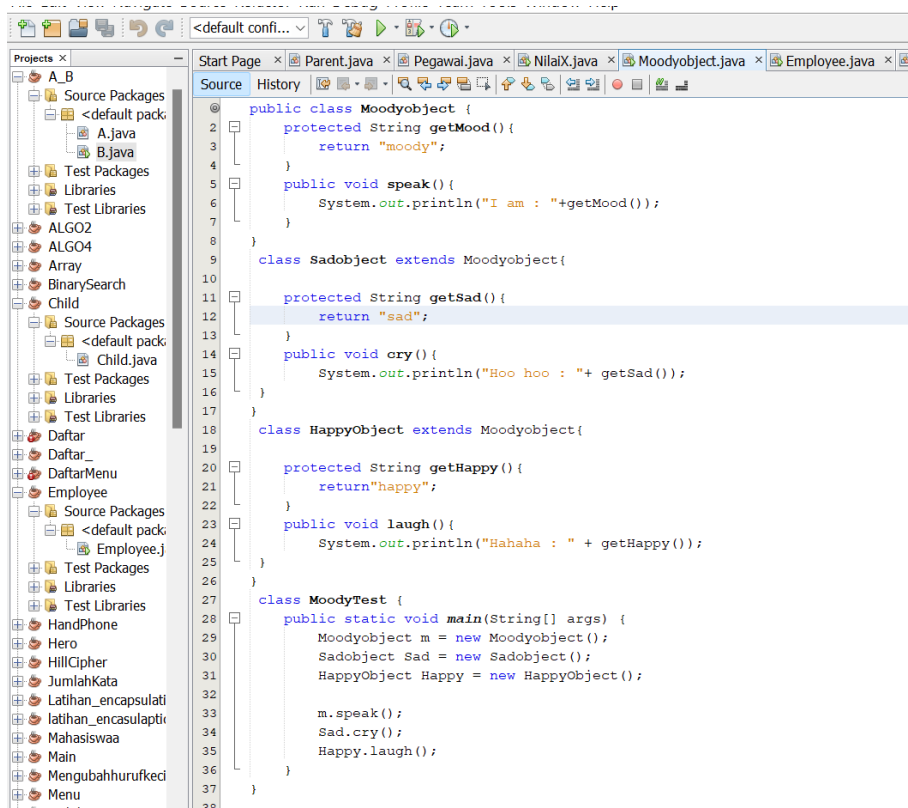
- o laugh, method untuk menampilkan pesan, tipe public

- MoodyObject berisi :

- o getMood, memberi nilai mood sekarang, tipe public, return type string

- o speak, menampilkan mood, tipe public.

INPUT



```
public class Moodyobject {
    protected String getMood(){
        return "moody";
    }

    public void speak(){
        System.out.println("I am : "+getMood());
    }
}

class Sadobject extends Moodyobject{

    protected String getSad(){
        return "sad";
    }

    public void cry(){
        System.out.println("Hoo hoo : " + getSad());
    }
}

class HappyObject extends Moodyobject{

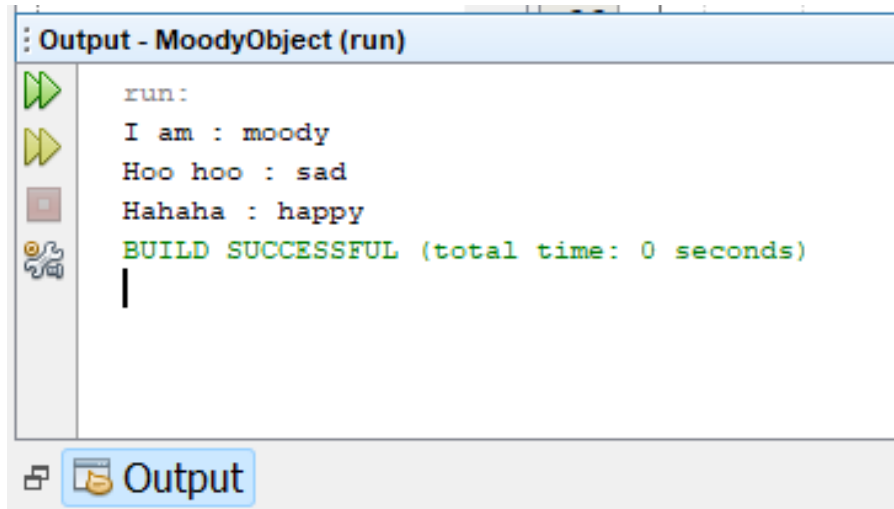
    protected String getHappy(){
        return "happy";
    }

    public void laugh(){
        System.out.println("Hahaha : " + getHappy());
    }
}

class MoodyTest {
    public static void main(String[] args) {
        Moodyobject m = new Moodyobject();
        Sadobject Sad = new Sadobject();
        HappyObject Happy = new HappyObject();

        m.speak();
        Sad.cry();
        Happy.laugh();
    }
}
```

OUTPUT



```
run:
I am : moody
Hoo hoo : sad
Hahaha : happy
BUILD SUCCESSFUL (total time: 0 seconds)
```

Percobaan diatas menunjukan penampilan pesan mood, dimana kelas ini diuji dengan moody test yang digunakan untuk menguji kelas dan subkelas. Mengapa public class di sub kelas dan di kelas moodytest dihapus publicnya, karena sudah merupakan tipe public jadi kita tidak perlu menuliskan lagi public di dalam subkelas dan kelas moodytest, hanya moodyobject. Jika ditambahkan public lagi pada sub kelas maka akan terjadi error dan tidak bisa di run.

Seperti ini:

```

Source Packages
<default pack
A.java
B.java
st Packages
braries
st Libraries
O2
O4
y
rySearch
j
ource Packages
<default pack
Child.java
st Packages
braries
st Libraries
ar
ar_
arMenu
loyee
ource Packages
<default pack
Employee.j
st Packages
braries
st Libraries
iPhone
,
ipher
lahKata
ian_encapsulati
an_encasulapti
siswaa
gubahhurufkeci
J
.

```

```

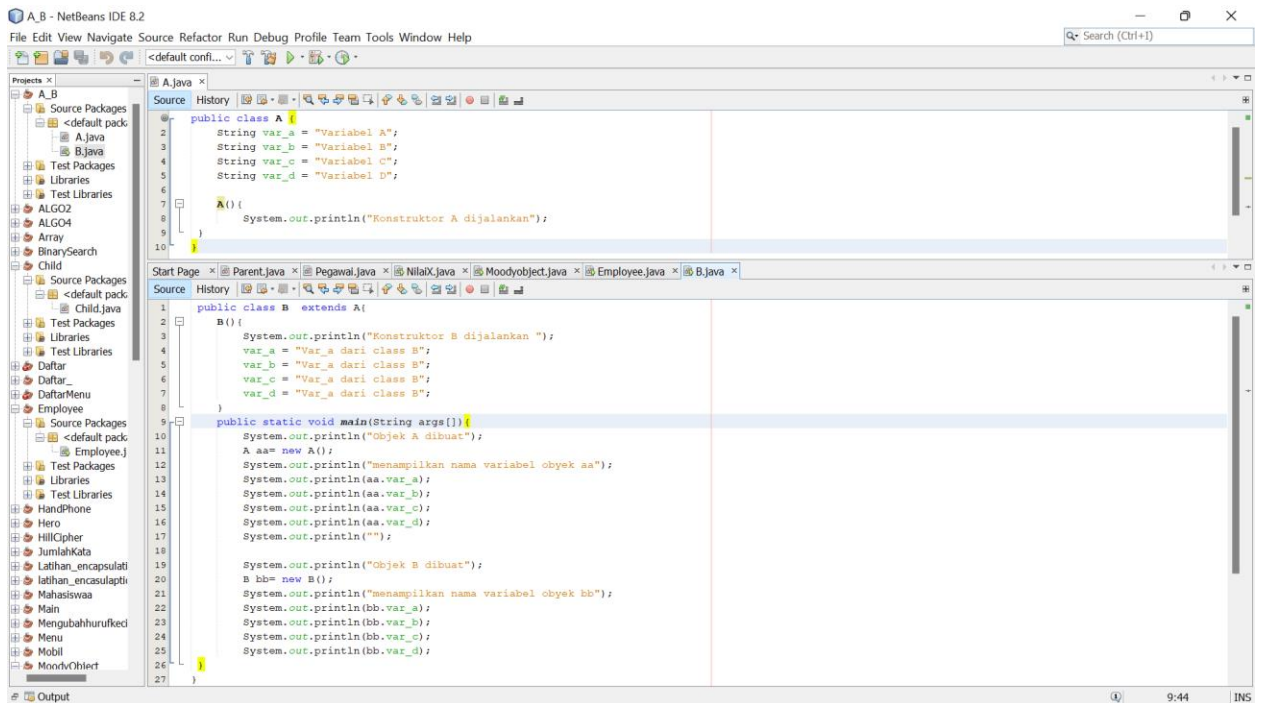
Start Page x Parent.java x Pegawai.java x NilaiX.java x Moodyobject.java x Em
Source History
1 public class Moodyobject {
2     protected String getMood() {
3         return "moody";
4     }
5     public void speak() {
6         System.out.println("I am : " + getMood());
7     }
8 }
9
10 public class Sadobject extends Moodyobject {
11     protected String getsad() {
12         return "sad";
13     }
14     public void cry() {
15         System.out.println("Hoo hoo : " + getSad());
16     }
17 }
18
19 public class HappyObject extends Moodyobject {
20     protected String getHappy() {
21         return "happy";
22     }
23     public void laugh() {
24         System.out.println("Hahaha : " + getHappy());
25     }
26 }
27
28 public class MoodyTest {
29     public static void main(String[] args) {
30         Moodyobject m = new Moodyobject();
31         Sadobject Sad = new Sadobject();
32         HappyObject Happy = new HappyObject();
33
34         m.speak();
35         Sad.cry();
36         Happy.laugh();
37     }
38 }

```

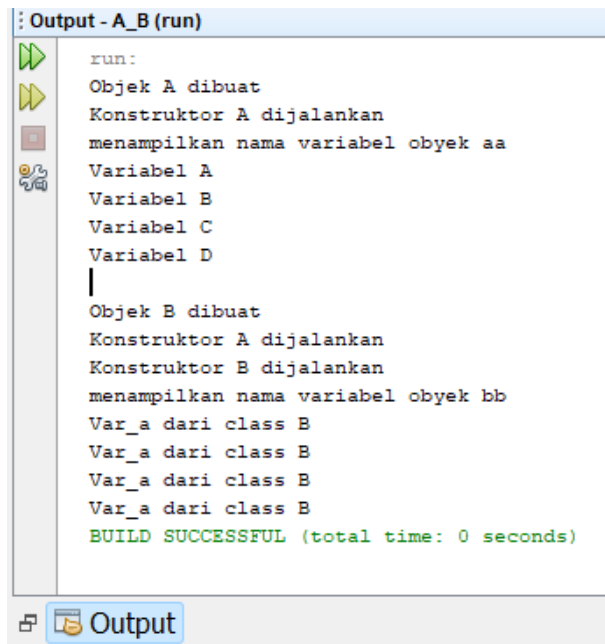
- PERCOBAAN 6
Percobaan berikut ini menunjukkan penggunaan kelas A dan dengan subkelas B. Simpan kedua kelas ini dalam 2 file yang berbeda (A.java dan B.java) dan dalam satu package. Perhatikan proses pemanggilan konstruktor dan pemanggilan variabel.

INPUT

Program A dan Program B dalam satu package A_B



OUTPUT

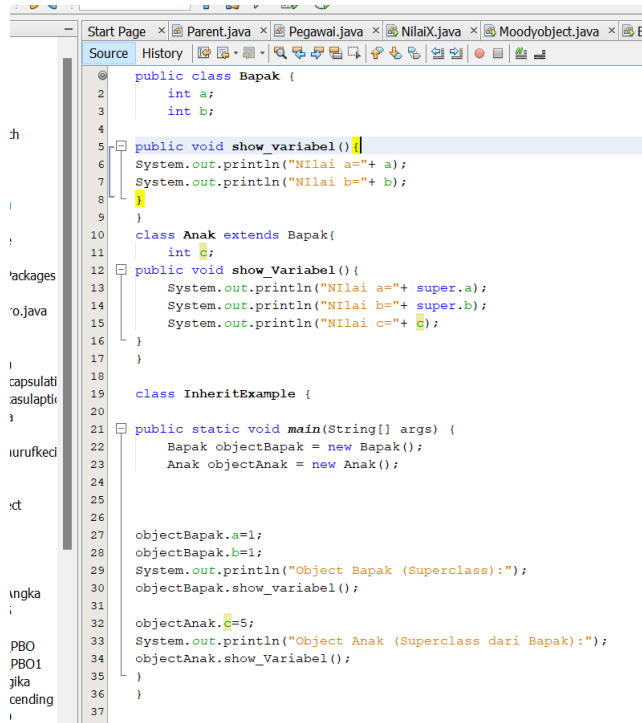


Percobaan diatas menggunakan kelas A dan sub kelas B dimana dalam 2 file yang berbeda di dalam satu package yang harus menghasilkan/ memanggil konstruktor dan memanggil variabel, kita hanya perlu menambahkan public disamping class karena berbeda file programnya kemudian menambahkan var_c dan var_d untuk pemanggilan konstruktor dan variabelnya.

- PERCOBAAN 7

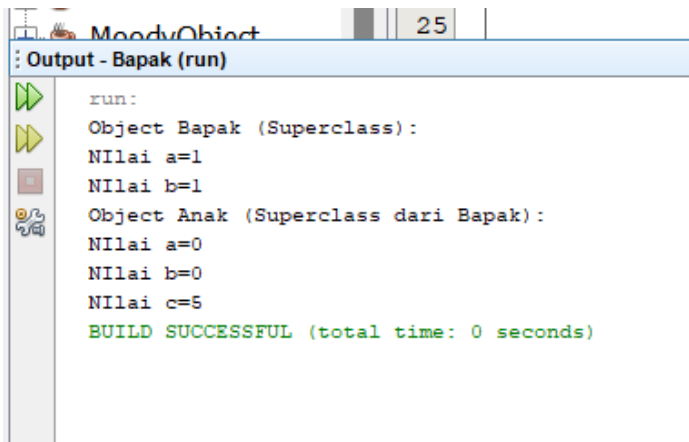
Percobaan berikut ini menunjukkan penggunaan Inheritance dan Overriding method pada kelas Bapak dan subkelas Anak. Terjadi override pada method show_variabel. Perhatikan perubahan nilai pada variabel a, b, dan c. Kemudian lakukan modifikasi pada method show_variabel() pada class Anak. Gunakan super untuk menampilkan nilai a dan b (memanfaatkan method yang sudah ada pada superclass).

INPUT



```
1 public class Bapak {
2     int a;
3     int b;
4
5     public void show_variabel() {
6         System.out.println("Nilai a=" + a);
7         System.out.println("Nilai b=" + b);
8     }
9
10    class Anak extends Bapak {
11        int c;
12        public void show_Variabel() {
13            System.out.println("Nilai a=" + super.a);
14            System.out.println("Nilai b=" + super.b);
15            System.out.println("Nilai c=" + c);
16        }
17    }
18
19    class InheritExample {
20
21        public static void main(String[] args) {
22            Bapak objectBapak = new Bapak();
23            Anak objectAnak = new Anak();
24
25
26
27            objectBapak.a=1;
28            objectBapak.b=1;
29            System.out.println("Object Bapak (Superclass):");
30            objectBapak.show_variabel();
31
32            objectAnak.c=5;
33            System.out.println("Object Anak (Superclass dari Bapak):");
34            objectAnak.show_Variabel();
35        }
36    }
37 }
```

OUTPUT



```
run:
Object Bapak (Superclass):
Nilai a=1
Nilai b=1
Object Anak (Superclass dari Bapak):
Nilai a=0
Nilai b=0
Nilai c=5
BUILD SUCCESSFUL (total time: 0 seconds)
```

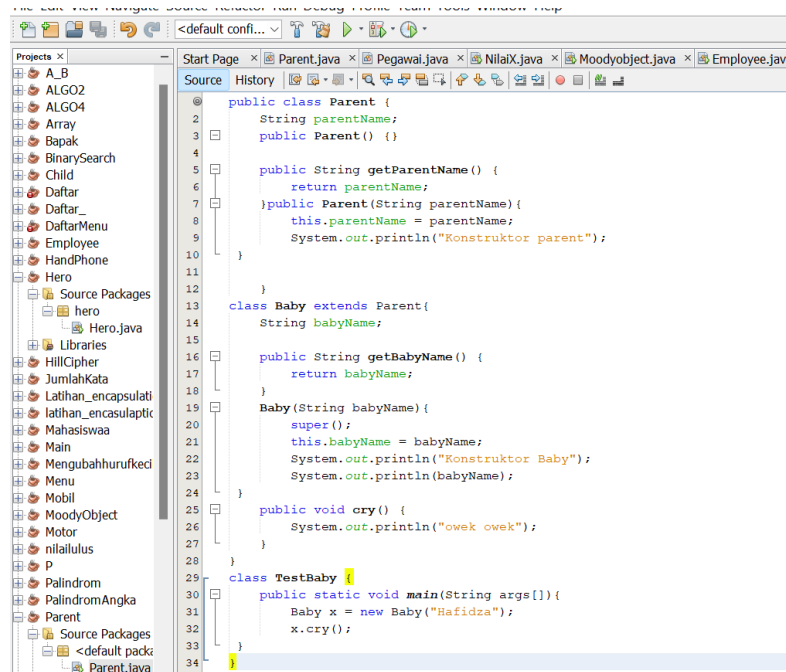
Percobaan diatas menampilkan bilangan a dan b juga perubahan variabel a,b,c yang telah di modifikasi di class anak. Mengapa class anak dan inheritExample tidak diberi public disamping

class, karena satu program dengan kelas bapak sehingga jika satu program diberikan public pada subclass akan terjadi error kecuali di berbeda program.

- PERCOBAAN 8

Percobaan berikut ini menunjukkan penggunaan overriding method pada kelas Parent dan subkelas Baby, saat dilakukan pemanggilan konstruktor superclass dengan menggunakan super.

INPUT



```
public class Parent {
    String parentName;
    public Parent() {}

    public String getParentName() {
        return parentName;
    }
    public Parent(String parentName) {
        this.parentName = parentName;
        System.out.println("Konstruktor parent");
    }
}

class Baby extends Parent {
    String babyName;

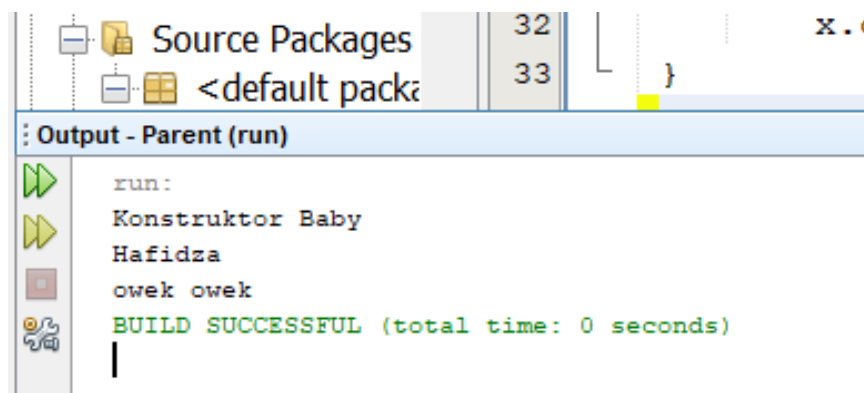
    public String getBabyName() {
        return babyName;
    }

    Baby(String babyName) {
        super();
        this.babyName = babyName;
        System.out.println("Konstruktor Baby");
        System.out.println(babyName);
    }

    public void cry() {
        System.out.println("owek owek");
    }
}

class TestBaby {
    public static void main(String args[]) {
        Baby x = new Baby("Hafidza");
        x.cry();
    }
}
```

OUTPUT



```
run:
Konstruktor Baby
Hafidza
owek owek
BUILD SUCCESSFUL (total time: 0 seconds)
```

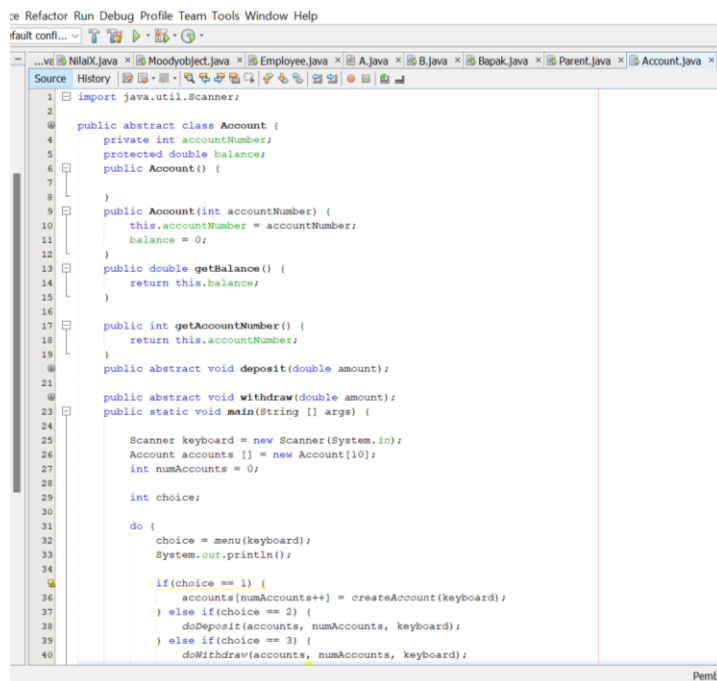
Percobaan diatas pemanggilan konstruktor dengan menggunakan super, dimana memanggil konstruktor Baby dengan nama hafidza dan menangis dengan kata owek owek berhasil dijalankan.

- PERCOBAAN 9

Pembuatan kelas Account dan subkelas SavingAccount, CheckingAccount

Buat kelas Account sesuai dengan diagram UML untuk kelas Account sebelumnya, dengan definisi :

- Atribut balance tipe double, dan sifat protected
- Constructor Account untuk memberi nilai awal balance
- Method getBalance untuk mendapatkan nilai balance
- Method deposit untuk menambah nilai balance
- Method withdraw untuk mengambil nilai balance



```
1 import java.util.Scanner;
2
3 public abstract class Account {
4     private int accountNumber;
5     protected double balance;
6     public Account() {
7     }
8
9     public Account(int accountNumber) {
10         this.accountNumber = accountNumber;
11         balance = 0;
12     }
13     public double getBalance() {
14         return this.balance;
15     }
16
17     public int getAccountNumber() {
18         return this.accountNumber;
19     }
20     public abstract void deposit(double amount);
21
22     public abstract void withdraw(double amount);
23     public static void main(String [] args) {
24
25         Scanner keyboard = new Scanner(System.in);
26         Account accounts [] = new Account[10];
27         int numAccounts = 0;
28
29         int choice;
30
31         do {
32             choice = menu(keyboard);
33             System.out.println();
34
35             if(choice == 1) {
36                 accounts[numAccounts++] = createAccount(keyboard);
37             } else if(choice == 2) {
38                 doDeposit(accounts, numAccounts, keyboard);
39             } else if(choice == 3) {
40                 doWithdraw(accounts, numAccounts, keyboard);
41             }
42         } while(choice != 0);
43     }
44 }
```

```
import java.util.Scanner;

public abstract class Account {
    private int accountNumber;
    protected double balance;
    public Account() {

    }
    public Account(int accountNumber) {
        this.accountNumber = accountNumber;
        balance = 0;
    }
}
```

```

public double getBalance() {
    return this.balance;
}

public int getAccountNumber() {
    return this.accountNumber;
}
public abstract void deposit(double amount);

public abstract void withdraw(double amount);
public static void main(String [] args) {

    Scanner keyboard = new Scanner(System.in);
    Account accounts [] = new Account[10];
    int numAccounts = 0;

    int choice;

    do {
        choice = menu(keyboard);
        System.out.println();

        if(choice == 1) {
            accounts[numAccounts++] =
createAccount(keyboard);
        } else if(choice == 2) {
            doDeposit(accounts, numAccounts, keyboard);
        } else if(choice == 3) {
            doWithdraw(accounts, numAccounts, keyboard);
        } else if(choice == 4) {
            applyInterest(accounts, numAccounts, keyboard);
        } else {
            System.out.println("GoodBye!");
        }
        System.out.println();
    } while(choice != 5);
}

```

```

public static int accountMenu(Scanner keyboard) {
    System.out.println("Select Account Type");
    System.out.println("1. Checking Account");
    System.out.println("2. Savings Account");

    int choice;
    do {
        System.out.print("Enter choice: ");
        choice = keyboard.nextInt();
    } while (choice < 1 || choice > 2);

    return choice;
}

public static int searchAccount(Account accounts [], int
count, int accountNumber) {

    for(int i=0; i<count; i++) {
        if(accounts[i].getAccountNumber() == accountNumber) {
            return i;
        }
    }

    return -1;
}

public static void doDeposit(Account accounts [], int count,
Scanner keyboard) {

    System.out.print("\nPlease enter account number: ");
    int accountNumber = keyboard.nextInt();

    int index = searchAccount(accounts, count,
accountNumber);

    if(index >= 0) {

```

```

        System.out.print("Please enter Deposit Amount: ");
        double amount = keyboard.nextDouble();

        accounts[index].deposit(amount);
    } else {
        System.out.println("No account exist with
AccountNumber: " + accountNumber);
    }
}

    public static void doWithdraw(Account accounts [], int count,
Scanner keyboard) {

        System.out.print("\nPlease enter account number: ");
        int accountNumber = keyboard.nextInt();

        int index = searchAccount(accounts, count,
accountNumber);

        if(index >= 0) {

            System.out.print("Please enter Withdraw Amount: ");
            double amount = keyboard.nextDouble();
            accounts[index].withdraw(amount);
        } else {
            System.out.println("No account exist with
AccountNumber: " + accountNumber);
        }
    }

    public static void applyInterest(Account accounts [], int
count, Scanner keyboard) {

        System.out.print("\nPlease enter account number: ");
        int accountNumber = keyboard.nextInt();

```

```

        int index = searchAccount(accounts, count,
accountNumber);

        if(index >= 0) {

            if(accounts[index] instanceof SavingAccount) {
                ((SavingAccount)accounts[index]).applyInterest();
            }
        } else {
            System.out.println("No account exist with
AccountNumber: " + accountNumber);
        }
    }

    public static Account createAccount(Scanner keyboard) {

        Account account = null;
        int choice = accountMenu(keyboard);

        int accountNumber;
        System.out.print("Enter Account Number: ");
        accountNumber = keyboard.nextInt();

        if(choice == 1) { // chekcing account
            System.out.print("Enter Transaction Fee: ");
            double fee = keyboard.nextDouble();
            account = new CheckingAccount(accountNumber, fee);
        } else { // Savings account

            System.out.print("Please enter Interest Rate: ");
            double ir = keyboard.nextDouble();
            account = new SavingAccount(accountNumber, ir);
        }
        return account;
    }

```

```
}

public static int menu(Scanner keyboard) {
    System.out.println("Bank Account Menu");
    System.out.println("1. Create New Account");
    System.out.println("2. Deposit Funds");
    System.out.println("3. Withdraw Funds");
    System.out.println("4. Apply Interest");
    System.out.println("5. Quit");

    int choice;

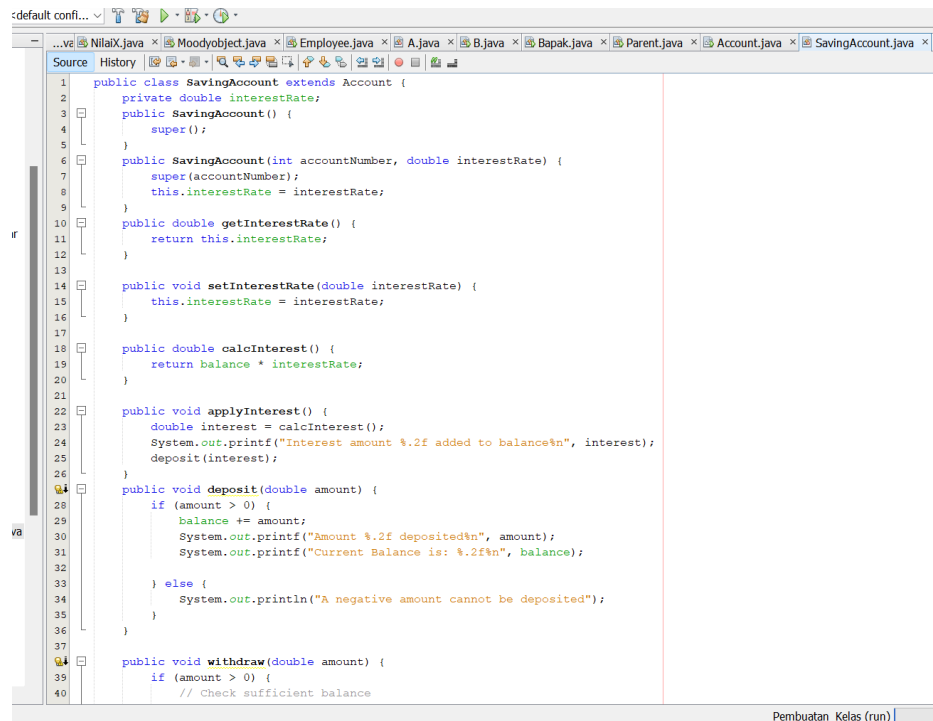
    do {
        System.out.print("Enter choice: ");
        choice = keyboard.nextInt();
    } while(choice < 1 || choice > 5);

    return choice;
}
}
```

Buat subkelas **SavingAccount** sesuai dengan diagram UML sebelumnya dengan definisi :

- Kelas **SavingAccount** merupakan turunan kelas **Account**, gunakan keyword **extends**.
- Atribut **interestRate**, tipe **double**, sifat **private**
- Constructor **SavingAccount** dengan parameter **balance** dan **interest_rate**.

Constructor ini harus passing parameter **balance** ke parent constructor dengan menggunakan **super(balance)** dan mengeset nilai variabel **interestRate** dengan nilai **interest_rate**.



The screenshot shows an IDE window with the following code for **SavingAccount.java**:

```
1 public class SavingAccount extends Account {
2     private double interestRate;
3     public SavingAccount() {
4         super();
5     }
6     public SavingAccount(int accountNumber, double interestRate) {
7         super(accountNumber);
8         this.interestRate = interestRate;
9     }
10    public double getInterestRate() {
11        return this.interestRate;
12    }
13
14    public void setInterestRate(double interestRate) {
15        this.interestRate = interestRate;
16    }
17
18    public double calcInterest() {
19        return balance * interestRate;
20    }
21
22    public void applyInterest() {
23        double interest = calcInterest();
24        System.out.printf("Interest amount %.2f added to balance\n", interest);
25        deposit(interest);
26    }
27
28    public void deposit(double amount) {
29        if (amount > 0) {
30            balance += amount;
31            System.out.printf("Amount %.2f deposited\n", amount);
32            System.out.printf("Current Balance is: %.2f\n", balance);
33        } else {
34            System.out.println("A negative amount cannot be deposited");
35        }
36    }
37
38    public void withdraw(double amount) {
39        if (amount > 0) {
40            // Check sufficient balance
```

```
public class SavingAccount extends Account {
    private double interestRate;
    public SavingAccount() {
        super();
    }
    public SavingAccount(int accountNumber, double interestRate)
    {
        super(accountNumber);
        this.interestRate = interestRate;
    }
    public double getInterestRate() {
        return this.interestRate;
    }
}
```



```
}

public void setInterestRate(double interestRate) {
    this.interestRate = interestRate;
}

public double calcInterest() {
    return balance * interestRate;
}

public void applyInterest() {
    double interest = calcInterest();
    System.out.printf("Interest amount %.2f added to
balance%n", interest);
    deposit(interest);
}

public void deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        System.out.printf("Amount %.2f deposited%n", amount);
        System.out.printf("Current Balance is: %.2f%n",
balance);
    } else {
        System.out.println("A negative amount cannot be
deposited");
    }
}

public void withdraw(double amount) {
    if (amount > 0) {
        // Check sufficient balance
        if ((amount) <= balance) {
            System.out.printf("Amount of %.2f withdrawn from
Account%n", amount);
            balance -= amount;
        }
    }
}
```

```

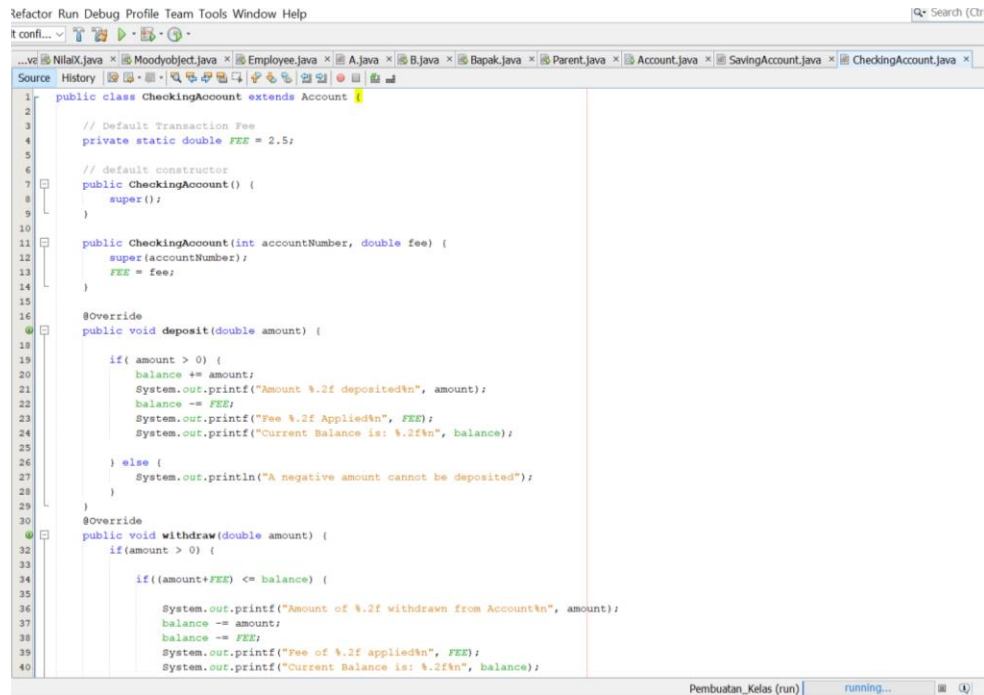
        System.out.printf("Current Balance is: %.2f%n",
balance);
    }
    } else {
        System.out.println("Negative amount cannot be
withdrawn!");
    }
}
}
}

```

Buat kelas CheckingAccount sesuai dengan diagram UML sebelumnya dengan definisi :

- Kelas CheckingAccount merupakan turunan kelas Account, gunakan keyword extends.
- Atribut overdraftProtection, tipe double, sifat private
- Terdapat public constructor dengan dua parameter: balance and protect. Constructor ini harus passing parameter balance ke parent constructor dengan menggunakan super(balance) dan mengeset nilai variabel overdraftProtection dengan nilai protect.
- Constructor dengan satu parameter yaitu balance. Constructor ini harus passing parameter balance ke lokal constructor dengan menggunakan this. Perhatikan bahwa constructor lain yang ada adalah constructor dengan dua parameter. Maka buat nilai protect default adalah -1.0 yang berarti bahwa pada account tidak terdapat overdraftProtection.
- Saldo = balance + overdraftProtection
- overdraftProtection = Saldo minimal, yaitu saldo yang diharapkan tidak boleh diambil pada suatu rekening, kecuali bila konsumen ingin menutup rekening.
- Class CheckingAccount harus mengoverride method withdraw. Method withdraw harus melakukan cek terhadap saldo (balance) apakah jumlahnya cukup bila terjadi pengambilan sejumlah uang (amount). Cek yang dilakukan adalah sebagai berikut :

- Jika $\text{balance} - \text{amount} \geq 0.0$ maka proses pengambilan diperbolehkan dan mengembalikan nilai true. Dan selanjutnya $\text{set balance} = \text{balance} - \text{amount}$;
- Jika $\text{balance} - \text{amount} < 0.0$ maka lakukan cek sebagai berikut:
- Jika tidak ada `overdraftProtection` (nilai = -1.0) atau $\text{overdraftProtection} < \text{overdraftNeeded} (\text{amount} - \text{balance})$ maka gagalkan proses pengambilan uang dengan mengembalikan nilai false.
- Jika terdapat `overdraftProtection` atau $\text{overdraftProtection} > \text{overdraftNeeded} (\text{amount} - \text{balance})$ maka proses pengambilan uang berhasil dengan mengembalikan nilai true. Dan selanjutnya $\text{set balance} = 0.0$; $\text{overdraftProtection} = \text{overdraftProtection} - \text{overdraftNeeded}$;
- Constructor dengan satu parameter yaitu `balance`. Constructor ini harus passing parameter `balance` ke lokal constructor dengan menggunakan `this`. Perhatikan bahwa constructor lain yang ada adalah constructor dengan dua parameter. Maka buat nilai `protect default` adalah -1.0 yang berarti bahwa pada account tidak terdapat `overdraftProtection`.
- $\text{Saldo} = \text{balance} + \text{overdraftProtection}$, `overdraftProtection` adalah saldo minimal, yaitu saldo yang diharapkan tidak boleh diambil pada suatu rekening, kecuali bila konsumen ingin menutup rekening.
- `Overdraft Protection (Proteksi Cerukan)` yaitu fasilitas kredit kepada nasabah penyimpan dana untuk menutupi cerukan; fasilitas kredit bank tersebut memungkinkan nasabah untuk menarik cek yang melebihi dana tersedia pada saldo akunya sehingga kelebihan penarikan dana tersebut dikenakan bunga harian; apabila kelebihan penarikan dana ditutup dengan fasilitas kreditnya, kelebihan penarikan itu tidak dikenakan bunga harian.



The screenshot shows an IDE window with the source code of the `CheckingAccount` class. The code is as follows:

```
1 public class CheckingAccount extends Account {
2
3     // Default Transaction Fee
4     private static double FEE = 2.5;
5
6     // default constructor
7     public CheckingAccount() {
8         super();
9     }
10
11     public CheckingAccount(int accountNumber, double fee) {
12         super(accountNumber);
13         FEE = fee;
14     }
15
16     @Override
17     public void deposit(double amount) {
18
19         if( amount > 0) {
20             balance += amount;
21             System.out.printf("Amount %.2f deposited\n", amount);
22             balance -= FEE;
23             System.out.printf("Fee %.2f Applied\n", FEE);
24             System.out.printf("Current Balance is: %.2f\n", balance);
25         } else {
26             System.out.println("A negative amount cannot be deposited");
27         }
28     }
29
30     @Override
31     public void withdraw(double amount) {
32         if(amount > 0) {
33
34             if((amount+FEE) <= balance) {
35
36                 System.out.printf("Amount of %.2f withdrawn from Account\n", amount);
37                 balance -= amount;
38                 balance -= FEE;
39                 System.out.printf("Fee of %.2f applied\n", FEE);
40                 System.out.printf("Current Balance is: %.2f\n", balance);
```

The IDE interface includes a menu bar (Refactor, Run, Debug, Profile, Team, Tools, Window, Help), a toolbar, and a tabbed editor showing several files. The status bar at the bottom indicates the current file is `Pembuatan_Kelas (run)` and it is in a `running...` state.

```
public class CheckingAccount extends Account {

    // Default Transaction Fee
    private static double FEE = 2.5;

    // default constructor
    public CheckingAccount() {
        super();
    }

    public CheckingAccount(int accountNumber, double fee) {
        super(accountNumber);
        FEE = fee;
    }

    @Override
    public void deposit(double amount) {

        if( amount > 0) {
            balance += amount;
```

```

        System.out.printf("Amount %.2f deposited\n", amount);
        balance -= FEE;
        System.out.printf("Fee %.2f Applied\n", FEE);
        System.out.printf("Current Balance is: %.2f\n",
balance);

    } else {
        System.out.println("A negative amount cannot be
deposited");
    }
}
@Override
public void withdraw(double amount) {
    if(amount > 0) {

        if((amount+FEE) <= balance) {

            System.out.printf("Amount of %.2f withdrawn from
Account\n", amount);
            balance -= amount;
            balance -= FEE;
            System.out.printf("Fee of %.2f applied\n", FEE);
            System.out.printf("Current Balance is: %.2f\n",
balance);

        }
    } else {
        System.out.println("Negative amount cannot be
withdrawn!");
    }
}
}

```

OUTPUT

The screenshot shows an IDE with a project named 'Pembuatan_Kelas' (run). The source code for 'SavingAccount.java' is displayed, showing a class that extends 'Account'. The code includes methods for creating a new account, setting interest rate, calculating interest, and applying interest. The output window shows the execution of a menu-driven program that creates a new account, selects a savings account, enters an account number, and a transaction fee.

```
1 public class SavingAccount extends Account {
2     private double interestRate;
3     public SavingAccount() {
4         super();
5     }
6     public SavingAccount(int accountNumber, double interestRate) {
7         super(accountNumber);
8         this.interestRate = interestRate;
9     }
10    public double getInterestRate() {
11        return this.interestRate;
12    }
13
14    public void setInterestRate(double interestRate) {
15        this.interestRate = interestRate;
16    }
17
18    public double calcInterest() {
19        return balance * interestRate;
20    }
21
22    public void applyInterest() {
```

run:

```
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice: 1

Select Account Type
1. Checking Account
2. Savings Account
Enter choice: 1
Enter Account Number: 2
Enter Transaction Fee: 500

Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice: |
```

Output Pembuatan_Kelas (run) running...