LAPORAN RESMI

Praktikum 8 Enkapsulasi

Mata Kuliah: Praktek Pemrograman Berbasis Objek

Disusun oleh:

M. Ainur Ramadhan (3122500047)

2 D3 Teknik Informatika B

Dosen Pengampu: Yanuar Risah Prayogi S.Kom., M.Kom.

PROGRAM STUDI D3 TEKNIK INFORMATIKA

POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

2023/2024

A. TUGAS PENDAHULUAN

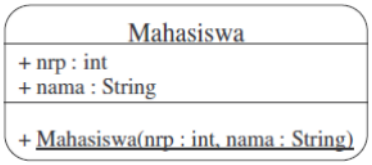
1. Apakah yang dimaksud dengan enkapsulasi?
   Jawab:
   Enkapsulasi adalah konsep dalam pemrograman berorientasi objek yang menggabungkan data dan metode yang beroperasi pada data ke dalam satu objek, dan umumnya melindungi data dari akses langsung dari luar objek. Ini memungkinkan pengendalian akses yang ketat terhadap data, abstraksi detail implementasi, dan memungkinkan perubahan internal yang aman. Dengan enkapsulasi, program dapat dirancang dengan lebih terstruktur, modular, dan dapat di-maintain, serta membantu dalam meningkatkan keamanan data.

2. Apakah yang dimaksud dengan constructor?
   Jawab:
   Constructor adalah metode khusus dalam pemrograman berorientasi objek (OOP) yang digunakan untuk menginisialisasi objek saat objek tersebut dibuat atau diinstansiasi dari sebuah kelas. Constructor memiliki nama yang sama dengan kelas tempatnya berada, dan mereka tidak mengembalikan nilai (tidak memiliki tipe kembalian).

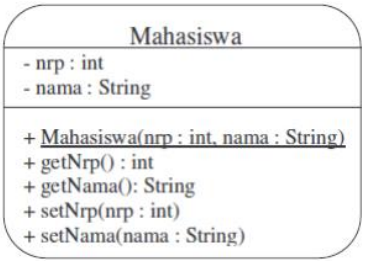
3. Apakah yang dimaksud dengan overloading constructor?
   Jawab:
   Overloading constructor (konstruktor overload) adalah konsep dalam pemrograman berorientasi objek di mana sebuah kelas dapat memiliki beberapa constructor dengan nama yang sama tetapi dengan daftar parameter yang berbeda. Dalam kata lain, kita dapat membuat beberapa versi constructor dalam kelas yang dapat menerima berbagai jenis dan jumlah argumen.

B. PERCOBAAN

Percobaan 1 : Melakukan enkapsulasi pada suatu class
Implementasikan UML class diagram Mahasiswa sebelum dan setelah dilakukan enkapsulasi!



Jika enkapsulasi dilakukan pada class diagram diatas, maka akan berubah menjadi:



Berikut implementasi kode program java untuk UML tersebut

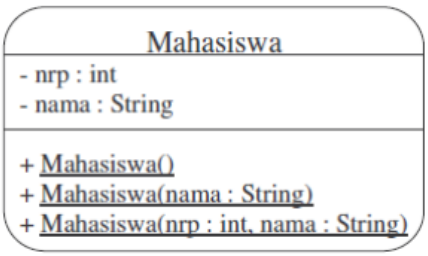
⇨ Sebelum diEnkapsulasi

```
public class Mahasiswa {
    private int nrp;
    private String nama;

    Mahasiswa(int nrp, String nama){
        this.nrp = nrp;
        this.nama = nama;
    }
}
```

⇨ Sesudah diEnkapsulasi

```
public class Mahasiswa {
    private int nrp;
    private String nama;

    Mahasiswa(int nrp, String nama){
        this.nrp = nrp;
        this.nama = nama;
    }

    public int getNrp() {
        return nrp;
    }

    public String getNama() {
        return nama;
    }

    public void setNrp(int nrp) {
        this.nrp = nrp;
    }

    public void setNama(String nama) {
        this.nama = nama;
    }
}
```

Percobaan 2 : Melakukan overloading constructor

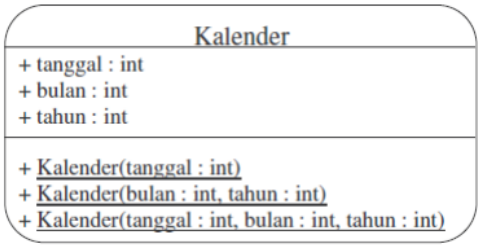


Dari class diagram tersebut, dapat diimplementasikan ke dalam program sebagai berikut:

```
public class Mahasiswa {
    public int nrp;
    public String nama;

    public Mahasiswa(){
        nrp= 0;
        nama= "";
    }

    public Mahasiswa(String nama){
        nrp= 0;
        this.nama = nama;
    }

    public Mahasiswa(int nrp, String nama){
        this.nrp= nrp;
        this.nama= nama;
    }
}
```

C. LATIHAN

Latihan 1 : Mengimplementasikan UML class diagram dalam program untuk class Kalender



Dari class diagram diatas, desainlah suatu class yang memenuhi konsep enkapsulasi. Untuk nilai inisialisasi, dipakai 1-1-2000. Pakailah kata kunci this untuk mempersingkat pengkodean. Tulislah listing program berikut ini sebagai pengetesan

```
public class TesKalender {
    public static String getTime(Kalender kal) {
        String tmp;
        tmp = kal.getTanggal() + "-" +
                kal.getBulan() + "-" +
                kal.getTahun();
        return tmp;
    }

    public static void main(String args[]) {
        Kalender kal = new Kalender(8);
        System.out.println("Waktu awal : " + getTime(kal));
        kal.setTanggal(9);
```

```
        System.out.println("1 hari setelah waktu awal : " + getTime(kal));
        kal = new Kalender(1,6,2003);
         System.out.println("Waktu berubah : " + getTime(kal));
        kal.setBulan(7);
        System.out.println("1 bulan setelah itu : " + getTime(kal));
        kal = new Kalender(20, 10, 2004);
        System.out.println("Waktu berubah : " + getTime(kal));
        kal.setTahun(2005);
        System.out.println("1 tahun setelah itu : " + getTime(kal));
    }
}
```

Berikut Kode Program untuk mengimplementasikan class Diagram Kalender

⇨ *Kalender.java*

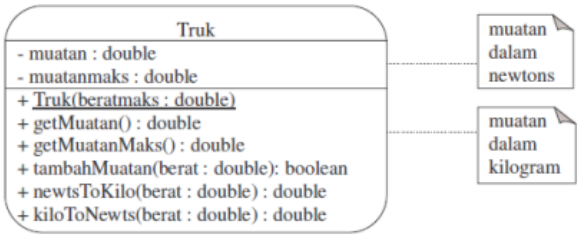
```
public class Kalender {
    private int tanggal=1;
    private int bulan=1;
    private int tahun=2000;

    public Kalender(int tanggal){
        bulan= 0;
        tahun= 0;
        this.tanggal= tanggal;
    }

    public Kalender(int bulan, int tahun){
        tanggal=0;
        this.bulan = bulan;
        this.tahun = tahun;
    }

    public Kalender(int tanggal, int bulan, int tahun){
        this.tanggal= tanggal;
        this.bulan= bulan;
        this.tahun= tahun;
    }

    public int getTanggal() {
        return tanggal;
    }

    public int getBulan() {
        return bulan;
    }

    public int getTahun() {
        return tahun;
```

```
    }

    public void setTanggal(int tanggal) {
        this.tanggal = tanggal;
    }

    public void setTahun(int tahun) {
        this.tahun = tahun;
    }

    public void setBulan(int bulan) {
        this.bulan = bulan;
    }

}
```

Output:

```
C:\Users\USER\OOP\Praktikum\Praktikum 8\Latihan\Latihan1>javac *.java

C:\Users\USER\OOP\Praktikum\Praktikum 8\Latihan\Latihan1>java TesKalender
Waktu awal : 8-1-2000
1 hari setelah waktu awal : 9-1-2000
Waktu berubah : 1-6-2003
1 bulan setelah itu : 1-7-2003
Waktu berubah : 20-10-2004
1 tahun setelah itu : 20-10-2005
```

Latihan 2 : Mengimplementasikan UML class diagram dalam program untuk class Truk



Keterangan : 1 kilogram = 9,8 newtons

Tranformasikan class diagram diatas ke dalam bentuk program! Tuliskan listing program berikut ini sebagai pengetesan.

```
public class TesTugas2 {
    public static void main(String args[]) {
        boolean status;
        Truk truk = new Truk(900);
        System.out.println("Muatan maksimal = " + truk.getMuatanMaks());
        status = truk.tambahMuatan(500.0);
        System.out.println("Tambah muatan : 500");
        if (status)
            System.out.println("Ok");
```

```
        else
            System.out.println("Gagal");
        status = truk.tambahMuatan(300.0);
        System.out.println("Tambah muatan : 300");
        if (status)
            System.out.println("Ok");
        else
            System.out.println("Gagal");
        status = truk.tambahMuatan(150.0);
        System.out.println("Tambah muatan : 150");
        if (status)
            System.out.println("Ok");
        else
            System.out.println("Gagal");
        status = truk.tambahMuatan(50.0);
        System.out.println("Tambah muatan : 50");
        if (status)
            System.out.println("Ok");
        else
            System.out.println("Gagal");
        System.out.println("Muatan sekarang = " + truk.getMuatan());
    }
}
```

Lakukan kompilasi pada program diatas dan jalankan. Jika tampilan di layar tampak seperti dibawah ini, maka program anda sudah benar. Jika tidak sama, benahi kembali program anda dan lakukan hal yang sama seperti diatas.

```
Muatan maksimal : 900.0
Tambah muatan : 500  ok
Tambah muatan : 300  ok
Tambah muatan : 150  gagal
Tambah muatan : 50  ok
Muatan sekarang = 849.9999999999999
```

Berikut implementasi kode program untuk UML diatas.

⇨ *Truk.java*

```
import java.text.DecimalFormat;
public class Truk {
    private double muatan=0;
    private double muatanMaks;

    public Truk(double beratMaks) {
        this.muatanMaks = kiloToNewts(beratMaks);
    }

    public double getMuatan() {
        return newtsToKilo(this.muatan);
```

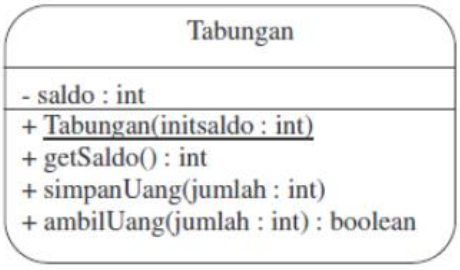
```
    }

    public double getMuatanMaks() {
        return Double.parseDouble(new
DecimalFormat("0.0").format(newtsToKilo(this.muatanMaks)));
    }

    public boolean tambahMuatan(double berat) {
        if (this.muatan + kiloToNewts(berat) <= this.muatanMaks) {
            this.muatan += kiloToNewts(berat);
            return true;
        } else {
            return false;
        }
    }

    public double kiloToNewts(double berat) {
        return berat * 9.8;
    }

    public double newtsToKilo(double berat) {
        return berat / 9.8;
    }
}
```

Output

```
C:\Users\USER\OOP\Praktikum\Praktikum 8\Latihan\Latihan2>java TesTugas2
Muatan maksimal = 900.0
Tambah muatan : 500
Ok
Tambah muatan : 300
Ok
Tambah muatan : 150
Gagal
Tambah muatan : 50
Ok
Muatan sekarang = 849.9999999999999
```

D. TUGAS

Tugas 1. Menerapkan konsep enkapsulasi pada kelas Tabungan yang terdapat di Tugas 1. Bab 7. Pengenalan Pemrograman Berbasis Obyek



Kembangkan kelas Tabungan diatas sehingga memungkinkan pengguna untuk memilih satuan mata uang yang berbeda (USD, AUD, IDR) ketika mengambil atau menyimpan uang. Saldo tabungan disimpan dalam satuan IDR oleh sistem. Beri nama kelas anda dengan nama MultiTabungan.java. Diasumsikan bahwa:
1 AUD = 10.000 IDR
1 USD = 9.000 IDR
Buat kelas baru untuk mengetes kelas MultiTabungan yang anda buat!

Berikut Source Code untuk kelas MultiTabungan

MultiTabungan.java

```java
package perbankan;

public class MultiTabungan {
    private double saldo;

    public MultiTabungan(double saldo) {
        this.saldo = saldo;
    }

    public MultiTabungan(double saldo, String mataUang) {
        this.saldo = konversiKeIDR(saldo, mataUang);
    }

    public double getSaldo() {
        return this.saldo;
    }

    public double getSaldo(String mataUang) {
        return konversiDariIDR(this.saldo, mataUang);
    }

    public void simpanUang(double jumlah) {
        this.saldo += jumlah;
    }

    public void simpanUang(double jumlah, String mataUang) {
        this.saldo += konversiKeIDR(jumlah, mataUang);
    }

    public boolean ambilUang(double jumlah) {
        if (this.saldo >= jumlah) {
            this.saldo -= jumlah;
            return true;
        }
        return false;
    }
```

```
    public boolean ambilUang(double jumlah, String mataUang) {
        double jumlahIDR = konversiKeIDR(jumlah, mataUang);
        if (this.saldo >= jumlahIDR) {
            this.saldo -= jumlahIDR;
            return true;
        }
        return false;
    }

    private double konversiKeIDR(double jumlah, String mataUang) {
        switch (mataUang) {
            case "USD":
                return usdToIdr(jumlah);
            case "AUD":
                return audToIdr(jumlah);
            default:
                return jumlah;
        }
    }

    private double konversiDariIDR(double jumlahIDR, String mataUang) {
        switch (mataUang) {
            case "USD":
                return idrToUsd(jumlahIDR);
            case "AUD":
                return idrToAud(jumlahIDR);
            default:
                return jumlahIDR;
        }
    }

    private double usdToIdr(double amount) {
        return amount * 9000;
    }

    private double idrToUsd(double amount) {
        return amount / 9000;
    }

    private double audToIdr(double amount) {
        return amount * 10000;
    }

    private double idrToAud(double amount) {
        return amount / 10000;
    }
}
```

Analisa Kode Program:
Pada Kelas MultiTabungan ini terdapat:

1. **Variabel Saldo**: Kelas ini memiliki sebuah variabel **saldo** yang digunakan untuk menyimpan jumlah saldo tabungan dalam bentuk IDR (Rupiah).
2. **Konstruktor**: Terdapat dua konstruktor, satu menerima saldo awal dalam IDR, dan yang lainnya menerima saldo awal dalam mata uang tertentu dan mengonversinya ke IDR.
3. **Metode getSaldo()**: Metode ini mengembalikan saldo tabungan dalam IDR.
4. **Metode getSaldo(String mataUang)**: Metode ini mengembalikan saldo tabungan dalam mata uang yang ditentukan dengan melakukan konversi dari IDR ke mata uang yang diminta.
5. **Metode simpanUang(double jumlah)**: Digunakan untuk menyimpan uang dalam bentuk IDR ke dalam saldo tabungan.
6. **Metode simpanUang(double jumlah, String mataUang)**: Digunakan untuk menyimpan uang dalam mata uang tertentu, yang kemudian diubah ke IDR sebelum ditambahkan ke saldo tabungan.
7. **Metode ambilUang(double jumlah)**: Digunakan untuk mengambil uang dari saldo tabungan dalam IDR jika saldo mencukupi. Mengembalikan **true** jika transaksi berhasil, dan **false** jika saldo tidak mencukupi.
8. **Metode ambilUang(double jumlah, String mataUang)**: Sama seperti **ambilUang(double jumlah)**, tetapi untuk mata uang tertentu, dengan konversi ke IDR terlebih dahulu.
9. **Metode Konversi Mata Uang**: Terdapat metode-metode private untuk melakukan konversi antara mata uang (contoh: USD ke IDR dan sebaliknya).

TesMultiTabungan.java

```
import perbankan.MultiTabungan;

public class TesMultiTabungan {
    public static void main(String[] args) {
        // Tanpa parameter mataUang
        System.out.println("== Tanpa Parameter mataUang ==");
        MultiTabungan tabunganA = new MultiTabungan(100000);
        System.out.println("Saldo Tabungan A: " + tabunganA.getSaldo());

        tabunganA.simpanUang(50000);
        tabunganA.ambilUang(120000);
        System.out.println("Saldo Tabungan A terbaru: " +
tabunganA.getSaldo());

        // Dengan parameter mataUang USD
        System.out.println("\n== Dengan Parameter mataUang (Tabungan B)
==");
        MultiTabungan tabunganB = new MultiTabungan(10, "USD");
        System.out.println("Saldo (USD): " + tabunganB.getSaldo("USD"));
        System.out.println("Saldo (AUD): " + tabunganB.getSaldo("AUD"));
        System.out.println("Saldo (IDR): " + tabunganB.getSaldo("IDR"));
```

```
        System.out.println("\n=> Simpan uang 5 USD");
        tabunganB.simpanUang(5, "USD");
        System.out.println("Saldo (USD): " + tabunganB.getSaldo("USD"));
        System.out.println("Saldo (AUD): " + tabunganB.getSaldo("AUD"));
        System.out.println("Saldo (IDR): " + tabunganB.getSaldo("IDR"));

        System.out.println("\n=> Simpan uang 3 AUD");
        tabunganB.simpanUang(3, "AUD");
        System.out.println("Saldo (USD): " + tabunganB.getSaldo("USD"));
        System.out.println("Saldo (AUD): " + tabunganB.getSaldo("AUD"));
        System.out.println("Saldo (IDR): " + tabunganB.getSaldo("IDR"));

        System.out.println("\n=> Simpan uang 20000 IDR");
        tabunganB.simpanUang(20000, "IDR");
        System.out.println("Saldo (USD): " + tabunganB.getSaldo("USD"));
        System.out.println("Saldo (AUD): " + tabunganB.getSaldo("AUD"));
        System.out.println("Saldo (IDR): " + tabunganB.getSaldo("IDR"));

        System.out.print("\n=> Ambil uang 6 USD ");
        boolean status = tabunganB.ambilUang(6, "USD");
        System.out.println(status ? "(OK)" : "(Gagal)");
        System.out.println("Saldo (USD): " + tabunganB.getSaldo("USD"));
        System.out.println("Saldo (AUD): " + tabunganB.getSaldo("AUD"));
        System.out.println("Saldo (IDR): " + tabunganB.getSaldo("IDR"));

        System.out.print("\n=> Ambil uang 10 AUD ");
        status = tabunganB.ambilUang(10, "AUD");
        System.out.println(status ? "(OK)" : "(Gagal)");
        System.out.println("Saldo (USD): " + tabunganB.getSaldo("USD"));
        System.out.println("Saldo (AUD): " + tabunganB.getSaldo("AUD"));
        System.out.println("Saldo (IDR): " + tabunganB.getSaldo("IDR"));

        System.out.print("\n=> Ambil uang 100000 IDR ");
        status = tabunganB.ambilUang(100000, "IDR");
        System.out.println(status ? "(OK)" : "(Gagal)");
        System.out.println("Saldo (USD): " + tabunganB.getSaldo("USD"));
        System.out.println("Saldo (AUD): " + tabunganB.getSaldo("AUD"));
        System.out.println("Saldo (IDR): " + tabunganB.getSaldo("IDR"));
    }
}
```

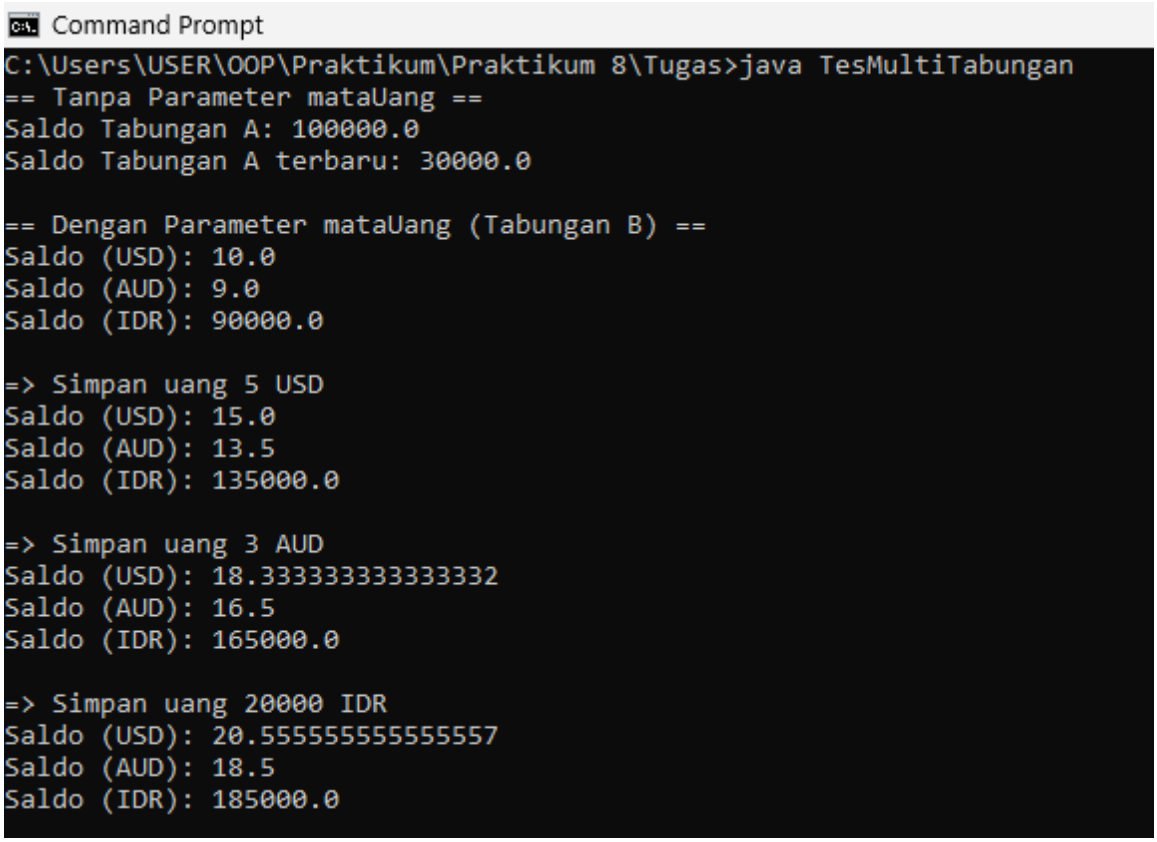
Analisa Kode Program:

1. **Import Kelas MultiTabungan**: Kode mengimpor kelas **MultiTabungan** dari paket **perbankan**, yang berarti kelas **TesMultiTabungan** dapat mengakses dan menggunakan kelas **MultiTabungan**.
2. **Metode main**: Metode **main** adalah titik awal eksekusi program. Dalam metode ini, terdapat beberapa pengujian terhadap kelas **MultiTabungan**.

1. **Pengujian Tanpa Parameter Mata Uang (Tabungan A)**:
    - Program Membuat objek **tabunganA** dengan saldo awal 100,000 IDR dan menampilkan saldo awalnya.
    - Program kemudian menyimpan 50,000 IDR dan mengambil 120,000 IDR dari tabunganA.
    - Terakhir, Program menampilkan saldo terbaru dari tabunganA.
2. **Pengujian Dengan Parameter Mata Uang (Tabungan B)**:
    - Program membuat objek **tabunganB** dengan saldo awal 10 USD dan menampilkan saldo dalam mata uang yang berbeda (USD, AUD, IDR).
    - Kemudian, Program melakukan operasi simpan dan ambil uang dalam berbagai mata uang (USD, AUD, IDR) dan menampilkan saldo terbaru setelah setiap operasi.

3. **Pesan Output**: Setiap langkah pengujian diikuti dengan pesan output yang menjelaskan tindakan yang diambil (seperti "Simpan uang 5 USD" atau "Ambil uang 6 USD"). Status operasi (berhasil atau gagal) juga dicetak.

Output



```
=> Ambil uang 6 USD (OK)
Saldo (USD): 14.555555555555555
Saldo (AUD): 13.1
Saldo (IDR): 131000.0

=> Ambil uang 10 AUD (OK)
Saldo (USD): 3.4444444444444446
Saldo (AUD): 3.1
Saldo (IDR): 31000.0

=> Ambil uang 100000 IDR (Gagal)
Saldo (USD): 3.4444444444444446
Saldo (AUD): 3.1
Saldo (IDR): 31000.0
```

E. KESIMPULAN

Enkapsulasi dalam pemrograman berorientasi objek adalah bahwa data dan metode yang beroperasi pada data tersebut dikemas bersama dalam satu unit yang disebut kelas. Ini membantu dalam pengendalian akses data dan mengabstraksi rincian implementasi. Konstruktor, sebagai metode khusus dalam kelas, digunakan untuk menginisialisasi objek dari kelas tersebut dengan nilai-nilai awal yang sesuai. Overloading konstruktor memungkinkan definisi beberapa konstruktor dengan parameter yang berbeda untuk fleksibilitas dalam pembuatan objek tanpa harus membuat kelas yang berbeda. Dengan kombinasi enkapsulasi dan konstruktor, pemrogram dapat mengendalikan akses data secara lebih ketat, mengamankan data, dan memberikan kemampuan untuk menginisialisasi objek dengan cara yang berbeda sesuai dengan kebutuhan.