1. Teori SDLC

Apa yang dimaksud dengan SDLC?

Software Development Life Cycle / SDLC: proses pengembangan software berdasarkan fase (umumnya dimulai dari *planning* hingga *maintenance*), dengan tujuan menjaga kualitas dan memastikan software sesuai dengan spesifikasi.

Keuntungan dari SDLC?

Salah satu fase SDLC adalah Fase Planning: fokus pada beberapa hal seperti "Bagaimana cara *tracking project progress*?", "*Schedule*", dan "estimasi waktu dan biaya yang dibutuhkan proyek".

1. Bagi departement Finance: hal ini memudahkan proses *budgeting* dan proses pembuatan antisipasi risiko kerugian finansial jika proyek gagal.
2. Bagi Lead Developer: memudahkan proses *tracking project progress* dan karena adanya laporan *progress*, memudahkan Lead Developer komunikasi dengan *stakeholders* lain.
3. Bagi Software Engineer: Pembagian tugas / fokus Software Engineer dari "Bagaimana cara menyelesaikan proyek ini?" menjadi "Bagaimana cara menyelesaikan 1 masalah / fitur ini?".

Sebutkan 3 model SDLC, serta kelebihan dan kekurangannya.

Agile Model:

Model ini melihat produk sebagai sebuah fitur / masalah yang harus diselesaikan. Project Manager akan memutuskan fitur mana yang akan diselesaikan terlebih dahulu. Fitur ini akan diselesaikan dalam periode waktu (umumnya 2 minggu), selama periode ini, *software engineer* hanya akan fokus menyelesaikan fitur yang telah ditentukan di awal periode.

Fokus Agile Model ini adalah "setiap fitur harus diselesaikan dalam periode waktu yang ditentukan dan harus sesuai dengan *feedback user*.*".*

Keuntungan Agile Model:

1. Fokus mencapai target: Setiap periode waktu selalu ada target yang harus dicapai (umumnya disebut Agile Board: "To Do", "In Progress", "Done"). Selama periode waktu, "To Do" tidak akan bertambah.
2. Target yang dibuat berdasarkan *feedback user.*
3. Agile Model sesuai untuk proyek yang *Software Development Requirements* nya belum *fixed* atau bisa berubah.

Kerugian Agile Model:

1. Bagi junior Software Engineer / karyawan baru perlu waktu untuk *adjustment* karena dokumentasi produk hasil Agile Model umumnya minim.
2. Bergantung dengan *feedback user*. Komunikasi yang baik dengan *user* menentukan kualitas produk yang dihasilkan.
3. Maintability: Akibat dokumentasi yang minim (karena umumnya tidak menjadi prioritas) selama proses Agile, fase maintenance sangat bergantung dengan Software Engineer yang mengerjakan fitur / masalah tersebut.

Waterfall Model:

Waterfall Model adalah model proyek tidak bisa berlanjut jika fase sebelumnya belum selesai. Hal ini karena fokus Waterfall Model adalah dokumentasi yang lengkap / memudahkan proses *maintenance*.

Keuntungan Waterfall Model:

1. Bagi departemen Finance: Adanya Project Planning yang detail dari awal hingga akhir, memudahkan proses *budgeting* dan membuat plan jika proyek gagal.
2. Proses pembagian tugas dari awal hingga akhir jelas.
3. Umumnya hasil produk dari Waterfall Model memiliki dokumentasi yang lengkap / proses *maintenance* tidak bergantung *Software Engineer* yang awalnya mengerjakan proyek tersebut.

Kerugian Waterfall Model:

1. Produk yang dihasilkan mungkin sudah tidak sesuai dengan yang dibutuhkan pasar (plan dengan kebutuhan pasar sudah berbeda seiring berjalannya waktu).
2. *Minimum Viable Product* memerlukan waktu yang lama sebelum produk bisa digunakan *user* / selama proses Waterfall, MVP hanya bisa digunakan *user* setelah prose *Deployment.*
3. Umumnya tidak bisa mengakomodir perubahan *Software Development Requirements.*

Iterative & Incremental Model:

Interative & Incremental Model adalah model proyek yang membagi produk menjadi sebuah MVP, berbeda dengan Agile Model, setiap MVP merupakan mini produk yang bisa digunakan oleh *user*.

Keuntungan Iterative & Incremental Model:

1. Karena fokus setiap proses Iterative & Incremental Model adalah membuat MVP / produk yang bisa digunakan *beta tester*. Proyek bisa melakukan *adjustment* jika menemukan *design flaw* sebelum proyek dirilis.
2. Karena tim yang dibutuhkan lebih besar, proses *Deployment* menjadi lebih cepat / *Time to market* lebih pendek.
3. Karena produk dibagi menjadi beberapa MVP, *tracking progress menjadi lebih mudah*.

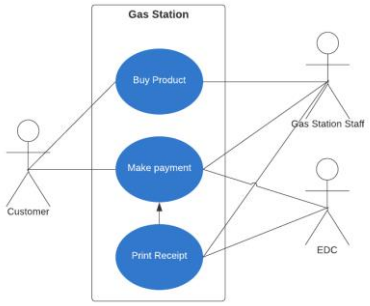
Kerugian Iterative & Incremental Model:

1. Memerlukan tim yang besar / perlu perhatian lebih banyak dari Manajemen untuk mengelola tim ini.
2. Karena memerlukan tim yang besar, jika ada perubahan *Software Development Requirements*, *budget* yang dibutuhkan untuk perubahan ini bisa sangat besar.
3. Tidak sesuai untuk proyek kecil.

2. **Buat UML untuk Pengisian Bahan Bakar di Tempat Pengisian Bahan bakar.**

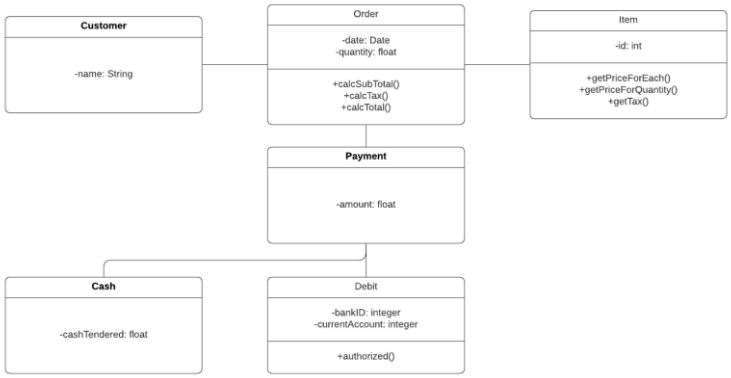
Scope analisa ini adalah "**Pengisian Bahan Bakar di Tempat Pengisian Bahan Bakar**", fitur lain yang mungkin dibutuhkan seperti "Pengelolahan Data Penjualan" atau "Pengisian Ulang Stok Bahan Bakar" tidak termasuk.

Use Case Diagram



Sistem Gas Station didesain berdasarkan tempat penggunaan nya, yaitu di sekitar mesin Gas Station. Sistem ini mendukung *print Receipt* melalui mesin Gas station dan mendukung pmebayaran debit melalui mesin EDC.

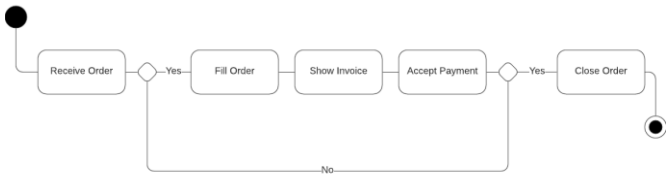
Class Diagram



Setiap kotak mewakili sebuah kotak, baris pertama adalah nama Class, baris kedua adalah attribute dan baris ketiga adalah method. Seluruh attribute menggunakan modifier "private" dengan tujuan mencegah attribute di class lain diubah oleh class lain. Dalam scope ini tidak ada method "Setters", hanya ada metode "Getters" untuk mendapatkan data dari class lain.

Umumnya class “Order” dibagi menjadi “Order” dan “OrderDetail”, untuk scope “Pengisian Bahan Bakar di Tempat Pengisian Bahan Bakar” tidak memerlukan class “OrderDetail” karena umumnya *customer* hanya membeli 1 jenis produk per transaksi (misalnya Bensin Pertamax x Liter). Scope ini tidak mendukung pembelian produk lain (makanan, dsb seperti Pom Bensin Shell x Deli2Go).

Activity Diagram



Activity “Receiver Order” dimulai ketika “Gas Station Staff” memasukan attribute “id" Item dan “quantity” Order, dan Activity “Close Order” ditandai dengan proses “Print Receipt”.

3. **Teori Object dan Class**

Apa yang dimaksud dengan Object, apakah yang dimaksud dengan Class? buatlah suatu Class sederhana dan buat Object dari kelas tersebut! (Buat Comment)

Object adalah hasil cetakan yang dibentuk menggunakan cetakan yang **sebelumnya sudah ditentukan**. Dalam Java, **Object tidak menyimpan *attribute*, dan *method, Object* menyimpan *Reference*** yang fungsi nya sebagai menunjukkan lokasi dimana *attribute* dan *method* tersebut disimpan di dalam *memory heap*. *Reference* ini memungkinkan kita untuk mengakses *attribute* dan juga *method* obyek tersebut.

Class adalah cetakan untuk membuat obyek tersebut.

```
class ObjectConstructor {
  // atribute
  private String name;
  private int age;

  // constructor to create a new Instance of Object.
  ObjectConstructor(String name, int age) {
    this.setName(name);
    this.setAge(age);
  }

  // method
  public String getName() {
    return name;
  }

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

```
  public int getAge() {
    return age;
  }

  public void setAge(int age) {
    this.age = age;
  }
}

public class ClassObject {
  public static void main(String[] args) {
    ObjectConstructor objectExample = new ObjectConstructor("Jason", 20); // This create new object with attributes.
    System.out.println("Name: " + objectExample.getName() + '\n' + "Age: " + objectExample.getAge()); // we can get the attributes using the method provided by the Object.
    System.out.println("Reference: " + objectExample); // Object stores Reference to where the attribute and method is stored in a memory heap.
  }
}
```

4. **Class Mobil**

```
public class Mobil {
  // attribute Mobil
  private int tahun;
  private String merek;
  private int kecepatan;

  // constructor untuk membuat Object Mobil
  Mobil(int tahun, String merek, int kecepatan) {
    this.tahun = tahun;
    this.merek = merek;
    this.kecepatan = kecepatan;
  }

  // method untuk menambah kecepatan
  void Tambah_Kecepatan(int kecepatan) {
    this.kecepatan += kecepatan;
    System.out.println("Menambah kecepatan. Kecepatan sekarang: " +
this.kecepatan);
  }

  // method untuk mengurangi kecepatan
  void Kurangi_Kecepatan(int kecepatan) {
    this.kecepatan -= kecepatan;
    System.out.println("Mengurangi kecepatan. Kecepatan sekarang: " +
this.kecepatan);
  }
```

```
  // method untuk sysout() attribute yang dimiliki oleh Object.
  void printStates() {
    System.out.println(
      "Tahun: " + this.tahun + '\n' +
      "Merek: " + this.merek + '\n' +
      "Kecepatan: " + this.kecepatan
    );
  }

  public static void main(String[] args) {
    Mobil mobil1 = new Mobil(2020, "Tesla", 0);
    mobil1.printStates(); // sysout() attribute yang dimiliki Object mobil1
    mobil1.Tambah_Kecepatan(10); // Setters untuk mengubah attribute Object
mobil1
    mobil1.Kurangi_Kecepatan(5); // Setters.
  }
}
```

5. **Class BangunRuang**

```
public class BangunRuang {
  int feature;
  float result;

  // Constructor Volume Balok
  BangunRuang(int feature, float panjang, float lebar, float tinggi) {
    if (feature == 1) {
      this.feature = feature;
      this.result = panjang * lebar * tinggi;
    } else {
      System.out.println("Invalid feature.");
    }
  }

  // Constructor Volume Bola, Volume Kubus
  BangunRuang(int feature, float value) {
    switch (feature) {
      // Volume Bola
      case 2:
        this.feature = feature;
        this.result = 22/7F * value * value * value * 4/3F;
        break;
```

```
    // Volume Kubus
    case 3:
      this.feature = feature;
      this.result = value * value * value;
      break;
    default:
      System.out.println("Invalid feature.");
  }
}

// method untuk menampilkan hasil perhitungan
void printState() {
  switch (this.feature) {
    case 1:
      System.out.println("Volume Balok: " + this.result);
      break;
    case 2:
      System.out.println("Volume Bola: " + this.result);
      break;
    case 3:
      System.out.println("Volume Kubus: " + this.result);
      break;

    default:
      System.out.println("Invalid feature.");
  }
}
```

```java
// method untuk menampilkan dari 3 nilai;
static void printStates(float...array) {
  System.out.println(
    "Volume Balok: " + array[0] + '\n' +
    "Volume Bola: " + array[1] + '\n' +
    "Volume Kubus: " + array[2] + '\n'
  );
}

// method untuk menghitung average
float getState() {
  return this.result;
}

static void calcAverage(float...array) {
  float result = 0;
  for (int index = 0; index < array.length; index++) {
    result += array[index];
  }
  result = result / array.length;
  System.out.println("Average: " + result);
}
```

```
  public static void main(String[] args) {

    System.out.println("Method getters State." + '\n');

    BangunRuang balok = new BangunRuang(1, 3, 4, 5);
    balok.printState();
    BangunRuang bola = new BangunRuang(2, 7);
    bola.printState();
    BangunRuang kubus = new BangunRuang(3, 3);
    kubus.printState();

    System.out.println('\n' + "Method getters Average." + '\n');

    // Memanggil method untuk menghitung summary berdasarkan parameter
    // yang diberikan.
    calcAverage(balok.getState(), bola.getState(), kubus.getState());

    System.out.println('\n' + "Method getters Summary." + '\n');

    // Memanggil method untuk menampilkan summary berdasarkan parameter
    // yang diberikan
    printStates(balok.getState(), bola.getState(), kubus.getState());
  }
}
```

6. **Class Array Multi Dimensi**

```
public class ArrayMultiDimensi {
  /**
   * Schema: [ [Nama, UTS, UAS, TUGAS], [Nama, UTS, UAS, TUGAS] ]
   */
  String[][] mahasiswa;

  // Constructor untuk membuat object.
  ArrayMultiDimensi(int size) {
    this.mahasiswa = new String[size][];
  }

  // method untuk tambah mahasiswa ke Object.
  void addMahasiswa(String nama, String uts, String uas, String tugas) {
    for (int index = 0; index < this.mahasiswa.length; index++) {
      if (this.mahasiswa[index] == null) {
        this.mahasiswa[index] = new String[] { nama, uts, uas, tugas };
        break;
      }
    }
  }

  // method to print data per row.
  void printState() {
    for (int index = 0; index < this.mahasiswa.length; index++) {
      String[] data = this.mahasiswa[index];
      float result = 0.35F * Float.parseFloat(data[1]) + 0.45F *
Float.parseFloat(data[2]) + 0.2F * Float.parseFloat(data[3]);
      System.out.println(
        "|  " + this.mahasiswa[index][0] + "  "+
        "|   " + this.mahasiswa[index][1] + "  " +
```

```
        "|   " + this.mahasiswa[index][2] + "  " +
        "|    " + this.mahasiswa[index][3] + "   " +
        "|      " + result + "     |"
      );
    }
  }

  // method to print the summary.
  void printSummary() {
    System.out.println("+--------+-------+-------+---------+--------------
+");
    System.out.println("|  Nama  |  UTS  |  UAS  |  TUGAS  |  NILAI AKHIR
|");
    System.out.println("+--------+-------+-------+---------+--------------
+");
    this.printState();
    System.out.println("+--------+-------+-------+---------+--------------
+");
  }

  public static void main(String[] args) {
    ArrayMultiDimensi mahasiswa = new ArrayMultiDimensi(3);
    // add data "mahasiswa" to the attribute.
    mahasiswa.addMahasiswa("Ardi", "60", "70", "90");
    mahasiswa.addMahasiswa("Pian", "80", "70", "90");
    mahasiswa.addMahasiswa("Robi", "70", "60", "90");

    // print the summary
    mahasiswa.printSummary();
  }
}
```