**LAPORAN IMPLEMENTASI PEMROGRAMAN BERORIENTASI OBJEK MENGGUNAKAN BAHASA *JAVA* DALAM PROGRAM *JOB PORTAL***

Disusun untuk Memenuhi Tugas Mata Kuliah Pemrograman Berorientasi Objek



Dosen Pengampu:

I Made Suartana, S.Kom., M.Kom.

Disusun Oleh:

Kelompok 6

Rayhan Ramadhani Hendra Atmadja 23051204075

Vena Novita Fitri Indriana 23051204077

Cornelius Louis Nathan 23051204085

PROGRAM STUDI (S1) TEKNIK INFORMATIKA  
 FAKULTAS TEKNIK  
 UNIVERSITAS NEGERI SURABAYA  
 2024

**KATA PENGANTAR**

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa. Karena atas limpahan rahmatnya kami dapat menyelesaikan laporan ini yang berjudul “LAPORAN IMPLEMENTASI PEMROGRAMAN BERORIENTASI OBJEK MENGGUNAKAN BAHASA *JAVA* DALAM PROGRAM *JOB PORTAL*” dengan tepat waktu tanpa adanya halangan yang berarti dan sesuai dengan harapan.

Adapun tujuan dari penulisan laporan ini adalah untuk memahami dan memperdalam bagaimana pengimplementasian Pemrograman Berorientasi Objek dengan menggunakkan bahasa *java* melalui program *job portal*. Pengimplementasiannya terdiri atas penggunaan *class* dan *object, inheritance, polymorph* dan *abstract*. Pembuatan laporan ini juga menjadi tugas yang mengisi nilai dari Mata Kuliah Pemrograman Berorientasi Objek.

Penyusunan makalah ini tidak lepas dari bantuan dan dukungan. Sehingga, kami menyampaikan terima kasih kepada pihak yang sudah mendukung pembuatan makalah ini. Terutama Bapak I Made Suartana, S.Kom., M.Kom. selaku dosen pengampu mata kuliah Pemrograman Berorientasi Objek yang telah membantu memberikan arahan dan pemahaman dalam penyusunan laporan ini.

Kami menyadari bahwa dalam penyusunan laporan ini masih banyak kekurangan karena keterbatasan kami. Maka dari itu penyusun sangat mengharapkan kritik dan saran untuk menyempurnakan laporan ini. Semoga apa yang ditulis dapat bermanfaat bagi semua pihak yang membutuhkan.

Surabaya, 25 Oktober 2024

                 Tertanda,

Penyusun

**DAFTAR ISI**

# BAB I

# PENDAHULUAN

## 1.1  Latar Belakang

Dalam *software engineering*, konsep Pemrograman Berorientasi Objek (PBO) memudahkan *developer* untuk membuat aplikasi yang terstruktur dan mudah dikelola. PBO atau OOP (*Object Oriented Programming*) memungkinkan pembuatan aplikasi menggunakkan objek yang mewakili entitas dan cara mereka berinteraksi. Sehingga, kode menjadi lebih modular, mudah untuk melakukan pemeliharaan. Sebagai contoh dalam pembuatan aplikasi *job portal*.

*Job Portal* adalah aplikasi untuk yang menghubungkan antara pencari kerja dan pemberi kerja. Aplikasi ini memudahkan pencari kerja untuk mencari dan melamar kerja sesuai dengan kriteria yang diinginkan dan memudahkan pemberi kerja untuk memposting dan mengelola iklan lowongan untuk menentukan kandidat yang cocok. Dengan banyaknya komponen yang dibutuhkan dalam pembuatan aplikasi ini,  OOP memungkinkan *developer* untuk membuat komponen tersebut menjadi objek atau *class* yang memudahkan pembuatan program, seperti pengguna, lowongan pekerjaan, dan perusahaan,

Selain itu, OOP mendukung konsep *inheritance*, *polymorphism*, dan *abstraction*, yang memungkinkan pengembang menambah fitur baru tanpa harus merubah kode yang sudah ada. Contohnya, jika kita memiliki class dasar *User*, kita mampu membuat class *Job Seeker* dan *Recruiter* dengan atribut serta metode yang sudah dikhususkan untuk setiap *class* tersebut. Ini memungkinkan *Job Portal* untuk melayani *user* dengan berbagai kebutuhan yang berbeda, apakah *user* ingin mengakses *job portal* sebagai *job seeker* atau *recruiter*.

Dengan menggunakan konsep OOP, pengembangan *Job Portal* menjadi lebih efisien dan responsif terhadap perubahan pasar. Keandalan dan kepraktisan sistem perawatan sangat krusial dalam lingkungan yang selalu berubah, di mana tren pekerjaan dan teknologi terus maju. Sehingga, menerapkan OOP dalam proyek portal lowongan kerja adalah langkah penting untuk membuat solusi yang sesuai dan dapat disesuaikan di masa yang akan datang.

## 1.2  Rumusan Masalah

1. Bagaimana deskripsi program *job portal* yang sudah dibuat?
2. Bagaimana pengimplementasian pemrograman berorientasi objek pada program *job portal?*

## 1.3  Tujuan

1. Mengerti bagaimana deskripsi program *job portal* yang sudah dibuat.
2. Mengerti pengimplementasian pemrograman berorientasi objek pada program *job portal.*

# BAB II

# LANDASAN TEORI

## 2.1 Pemrograman Berorientasi Objek

*Object Oriented Programming* (OOP) atau dalam bahasa Indonesia Pemrograman Berorientasi Objek adalah suatu paradigma pemrograman berdasarkan konsep objek atau benda yang bisa mengandung data dan kode. Data dalam bentuk *block* (sering dikenal sebagai atribut atau *property*), dan kode dalam bentuk prosedur (sering dikenal sebagai *method*). Di OOP, program komputer dirancang dengan membuatnya dari objek yang berinteraksi satu sama lain.

Terdapat banyak sekali bahasa pemrograman di dunia ini yang mengimplementasikan OOP di dalamnya (seperti C++, Java, dan Python), bahasa - bahasa pemrograman tersebut juga mendukung pemrograman berorientasi objek ke tingkat yang lebih besar atau lebih kecil, biasanya OOP dikombinasikan dengan beberapa paradigma pemrograman lainnya, seperti pemrograman imperatif, pemrograman prosedural dan pemrograman fungsional.’

Pemrograman berorientasi objek menggunakan objek, tetapi tidak semua teknik dan struktur terkait didukung langsung dalam bahasa yang mendukung OOP.

Fitur - fitur umum yang dimiliki oleh OOP antara lain :

1. Objek / *Object* : Sebuah objek adalah sebuah struktur atau tipe data abstrak yang mengandung  *fields* (*variable* yang berisi data) dan *methods* (*subroutines* atau prosedur yang mendefinisikan perilaku sebuah objek dalam kode).
2. Pewarisan / *Inheritance* : Pada umumnya, pemrograman yang mengadopsi OOP memberikan akses untuk *inheritance* sebagai *reusability* kode dan *extensibility* antara bentuk *class* ataupun *prototype*. Bentuk - bentuk *inheritance* ini sangat berbeda, tetapi terminologi analog digunakan untuk mendefinisikan konsep objek dan *instance*.
3. Enkapsulasi / *Encapsulation* : Enkapsulasi mencegah kode eksternal agar tidak berhubungan dengan cara kerja internal suatu objek. Hal ini juga memfasilitasi *code refactoring*, sebagai contoh : memperbolehkan *author* dari *class* tersebut untuk merubah bagaimana objek-objek dari *class* tersebut merepresentasikan data nya secara internal tanpa khawatir terhadap segala perubahan yang terjadi di kode eksternal.
4. *Polymorphism* : *Subtype* – bentuk polimorfisme adalah ketika suatu kode dapat dipanggil secara independen dari *class* manapun selama masih dalam hierarki yang didukung. Sebagai contoh, objek dari tipe *Circle* dan *Square* berasal *class* umum yang disebut *Shape*. Fungsi *draw* untuk setiap untuk setiap jenis *Shape* mengimplementasikan apa yang diperlukan.

## 2.2 Bahasa Pemrograman *Java*

*Java* adalah bahasa pemrograman yang banyak digunakan untuk pengkodean aplikasi web. Bahasa pemrograman ini telah menjadi pilihan populer di kalangan pengembang selama lebih dari dua dekade, dengan jutaan aplikasi Java yang digunakan saat ini. Java adalah bahasa multi-platform, berorientasi objek, dan berpusat pada jaringan yang dapat digunakan sebagai platform itu sendiri. Java adalah bahasa pemrograman yang cepat, aman, dan dapat diandalkan untuk mengkodekan segala sesuatu mulai dari aplikasi seluler dan perangkat lunak perusahaan hingga aplikasi data besar dan teknologi sisi server.

Bahasa pemrogramman Java populer dalam banyak bidang, diantara lain :

1. *Game Development*

Banyak game mobile, komputer, dan video game populer dibuat di Java. Bahkan game modern yang mengintegrasikan teknologi canggih seperti pembelajaran mesin atau realitas virtual dibangun dengan teknologi Java.

1. *Cloud Computing*

Java sering disebut sebagai WORA - Write Once and Run Anywhere, sehingga cocok untuk aplikasi berbasis cloud yang terdesentralisasi. Penyedia layanan cloud memilih bahasa Java untuk menjalankan program di berbagai platform yang mendasarinya.

1. *Big Data*

Java digunakan untuk mesin pengolah data yang dapat bekerja dengan kumpulan data yang kompleks dan data real-time dalam jumlah besar.

1. Kecerdasan Buatan / *Artificial Intelligence*

Java adalah pembangkit tenaga listrik dari perpustakaan pembelajaran mesin. Stabilitas dan kecepatannya membuatnya sempurna untuk pengembangan aplikasi kecerdasan buatan seperti pemrosesan bahasa alami dan pembelajaran mendalam.

1. *Internet of Things*

Java telah digunakan untuk memprogram sensor dan perangkat keras pada perangkat edge yang dapat terhubung secara mandiri ke internet.

# BAB III

# PEMBAHASAN

## 3.1 Penjelasan Deskripsi Program

Program *Job Portal* adalah aplikasi portal kerja sederhana dimana pengguna dapat mendaftar sebagai pelamar (*Job Seeker*) atau perekrut (*Recruiter*). Untuk bahasa pemrograman yang digunakan adalah bahasa pemrograman *java* dan menggunakan beberapa struktur data untuk menyimpan data atau informasi pengguna.

Dimana ketika pengguna membuka aplikasi, pengguna akan diarahkan atau disajikan dengan 3 pilihan menu yaitu *register* (daftar), *login* (masuk), dan *exit* (keluar) dalam method *home()*. Ketika pengguna memilih untuk melakukan *register* , pengguna akan diminta untuk memilih sebagai pelamar atau sebagai perekrut. Untuk proses pendaftaran akun memerlukan informasi seperti nama, *email*, nomor telepon, dan alamat. ketika mendaftar sebagai perekrut, sistem akan membuat *user ID* secara otomatis menggunakan UUID.

Setelah melakukan register , pengguna bisa kembali pada menu utama dan masuk ke akun mereka. Namun, ketika pengguna ingin memasuki akun mereka, pengguna wajib untuk mengisi email yang telah terdaftar sebelumnya. Jika *email* telah terdaftar pada proses *register* sebelumnya, maka program akan memeriksa jenis akun pengguna menggunakkan *hashmap* dan program secara otomatis akan mengarahkan pengguna pada menu yang sesuai. Menu program terbagi menjadi dua *method*, yaitu menuJobSeeker() untuk pencari lowongan pekerjaan / *job seeker* dan menuRecruiter()untuk HRD / *recruiter*.

Dalam menuJobSeeker(), terdapat 5 *method* yang mencakup fungsi untuk melihat dan memperbarui profil pengguna, mengunggah *resume*, dan melamar pekerjaan yang telah diposting oleh *Recruiter,* serta *logout* dari akun terkait.

Sementara itu, menuRecruiter() memiliki 6 *method* yang mencakup fungsi untuk melihat profil pengguna, memposting lowongan pekerjaan, menyediakan lapangan kerja. selain itu *Recruiter* juga dapat menghapus lowongan pekerjaan sebelumnya dan melihat daftar pekerjaan yang telah diposting. Recruiter juga dapat melihat profil dan resume dari *JobSeeker*, sehingga mereka dapat memeriksa pengalaman kerja serta keterampilan yang dimiliki *JobSeeker*. Kemudian dapat memberi tahu kepada *JobSeeker* ketika *resume* mereka telah diterima atau ditolak, dan menginformasikan kepada *JobSeeker* untuk dilakukan  interview ketika diterima.

Dalam program *job portal* kami, kami menggunakan 8 *class* yang saling berhubungan satu dengan yang lain untuk membantu berjalannya proses aplikasi. *Class* tersebut terdiri atas:

1. Class *User*

Class *user* merupakan *abstract class* yang berfungsi sebagai blueprint untuk dua *class user* yaitu *JobSeeker* dan *Recruiter*. Dalam class user ini terdapat atribut yang wajib untuk diisi yaitu *userId, name, email, phone,* dan *address*.

Ketika objek pada *subclass* dibuat, *constructor* akan terus memastikan semua atribut sudah terisi. Hal ini sangat penting, agar setiap pengguna dapat memiliki informasi yang lengkap sebelum pengguna menggunakan aplikasi. *userID* ini menggunakkan *class* UUID untuk dapat menghasilkan ID unik untuk tiap pengguna. Untuk memudahkan *login*, kami menggunakkan *hashmap* dengan *string email*. Sehingga ketika login menggunakkan *email*, maka pengguna akan langsung terhubung dengan *object*, dalam kasus ini adalah akun, yang sudah dibuat.

|  |
| --- |
| public abstract class User {      private String userId;      private String name;      private String email;      private String phone;      private String address;      public User(String name, String email, String phone, String address) {          this.userId = UUID.randomUUID().toString(); // Menghasilkan ID unik          this.name = name;          this.email = email;          this.phone = phone;          this.address = address;      }      // Getter methods      public String getUserId() {          return userId;      }      public String getName() {          return name;      }      public String getEmail() {          return email;      }      public String getPhone() {          return phone;      }      public String getAddress() {          return address;      }  } |

1. Class *JobSeeker*

*Class JobSeeker* merupakan subclass dari user, dimana *Jobseeker* berperan sebagai pelamar pekerjaan. *JobSeeker* memiliki atribut tambahan yakni *class* resume, yang digunakan untuk menyimpan informasi riwayat pendidikan dan pengalaman kerja *JobSeeker*. *JobSeeker* wajib untuk mengisi *resume* sebelum melamar pekerjaan.

Dengan ini program dapat menyimpan sebuah informasi yang penting dan memudahkan pengguna untuk merubah atau mengelola resume mereka. *JobSeeker* dapat memilih lowongan pekerjaan yang diinginkan atau diminati.

|  |
| --- |
| public class JobSeeker extends User {  private string Resume;   private String jobApplied;   private String preferences;   public JobSeeker(String name, String email, String phone, String address) {          super(name, email, phone, address);          this.resume = new Resume(); // Membuat objek Resume baru  }  public Resume getResume() {          return resume;  }  public void setResume(Resume resume) {          this.resume = resume;  }      // Metode untuk melamar pekerjaan  public void applyForJob(Job job) {  // Implementasi untuk melamar pekerjaan  }  } |

1. Class *Recruiter*

Sama seperti *JobSeeker, recruiter* juga merupakan *subclass* dari user, yang berperan sebagai perekrut atau perwakilan perusahaan yang memposting lowongan pekerjaan.  *Recruiter* memiliki atribut tambahan yaitu *companyId*, dimana *companyId* adalah identifikasi unik untuk perusahaan asal *recruiter* bekerja.

*Recruiter* untuk mencari pelamar atau kandidat yang memenuhi kriteria atau syarat pada posisi yang dibutuhkan oleh perusahaan. *Recruiter* dapat menambah pekerjaan baru, menghapus pekerjaan yang sudah tidak relevan atau dibutuhkan lagi, dan juga dapat melihat daftar pekerjaan yang telah diposting.

|  |
| --- |
| public class Recruiter extends User {  private String companyId;  public Recruiter(String name, String email, String phone, String address, String companyId) {           super(name, email, phone, address);          this.companyId = companyId;       }      public String getCompanyId() {           return companyId;       }      // Metode untuk memposting pekerjaan      public void postJob(Job job) {           // Implementasi untuk memposting pekerjaan       }  } |

1. Class *Resume*

*Class* yang terakhir yaitu *class resume* yang dapat mengolah semua informasi resume *JobSeeker*. Class resume ini memiliki atribut seperti resumeId, userId, lastEducation, dan workingExperiences untuk menyimpan suatu pengalaman kerja pelamar. *JobSeeker* dapat memperbarui *resume* dan dapat memastikan bahwa semua informasi sudah tersedia ketika ingin melamar suatu pekerjaan, serta *Recruiter* juga dapat mengakses *resume* untuk mengetahui apakah kandidat tersebut pantas untuk melamar di pekerjaan tersebut.

|  |
| --- |
| public class Resume {      private String resumeId;      private String userId;      private String lastEducation;      private ArrayList<Experience> workingExperiences;      public Resume() {          this.resumeId = UUID.randomUUID().toString(); // ID unik untuk resume          this.workingExperiences = new ArrayList<>();      }      public String getResumeId() {          return resumeId;      }      public String getUserId() {          return userId;      }      public void setUserId(String userId) {          this.userId = userId;      }      public String getLastEducation() {          return lastEducation;      }      public void setLastEducation(String lastEducation) {          this.lastEducation = lastEducation;      }      public ArrayList<Experience> getWorkingExperiences() {          return workingExperiences;      }      public void addWorkingExperience(Experience experience) {          this.workingExperiences.add(experience);      }  } |

1. Class *Job*

*Class Job* adalah *class* lowongan pekerjaan yang memiliki atribut yang terdiri atas JobId, JobName, jobDescription, atas recruitmentId. Selain itu terdapat juga atribut jobCategories dimana *job* ini dapat mengelompokkan suatu pekerjaan berdasarkan kategori tertentu dengan menggunakan *array*. Atribut ini sangat penting karena dapat membantu pelamar dalam menemukan lowongan pekerjaan yang sesuai dengan keahlian dan minat *JobSeeker*, serta membantu *Recruiter* dalam mencari . Atribut ini dibuat ketika semua objek sudah dibuat, sehingga data tentang pekerjaan yang sudah diposting selalu lengkap dan terstruktur.  JobId menggunakkan *class* UUID untuk dapat menghasilkan ID unik untuk tiap pengguna

|  |
| --- |
| public class Job {      private String jobId;      private String jobName;      private String jobDescription;      private String recruitmentId;      private String[] jobCategories;      public Job(String jobName, String jobDescription, String recruitmentId) {          this.jobId = UUID.randomUUID().toString(); // ID unik untuk pekerjaan          this.jobName = jobName;          this.jobDescription = jobDescription;          this.recruitmentId = recruitmentId;      }      // Getter methods      public String getJobId() {          return jobId;      }      public String getJobName() {          return jobName;      }      public String getJobDescription() {          return jobDescription;      }      public String getRecruitmentId() {          return recruitmentId;      }      public String[] getJobCategories() {          return jobCategories;      }      // Metode untuk menambahkan kategori      public void setJobCategories(String[] categories) {          this.jobCategories = categories;      }  } |

1. Class *Company*

*Class Company* ini juga memegang peran yang penting dalam sistem ini. Dimana *class* ini dapat mempresentasikan perusahaan dengan atribut name, address, phone, dan category. *Class* ini menjelaskan kepada *JobSeeker* tentang informasi perusahaan yang menawarkan lowongan pekerjaan terkait. Sehingga pengguna dapat mengetahui informasi lebih lanjut mengenai tempat mereka akan bekerja.

|  |
| --- |
| public class Company {      private String name;      private String address;      private String phone;      private String category;      public Company(String name, String address, String phone, String category) {          this.name = name;          this.address = address;          this.phone = phone;          this.category = category;      }      // Getter methods      public String getName() {          return name;      }      public String getAddress() {          return address;      }      public String getPhone() {          return phone;      }      public String getCategory() {          return category;      }  } |

1. Class *Application*

*Class Application* digunakan untuk menyimpan data lamaran pekerjaandan mengatur suatu proses lamaran pekerjaan. *Class Application* memiliki atribut yaitu applicationId, jobName, applicantName, applicationDate, yaitu status. *Class* ini membantu pengguna untuk memberitahu sampai mmana proses lamaran bekerja, seperti pengiriman CV apakah sudah diterima atau belum, proses interview atau wawancara, teknikal tes, status penerimaan, dan notes. Untuk status dari *class Application* menggunakkan tipe data *enum* untuk memudahkan pemberitahuan status, contohnya adalah APPLIED, REJECTED, SCHEDULED\_INTERVIEW,INTERVIEW\_COMPLETE, OFFERED, ACCEPTED , DECLINED.

|  |
| --- |
| public class Application {      private String applicationId;      private String jobName;      private String applicantName;      private String applicationDate;      private String status;      public Application(String jobName, String applicantName) {          this.applicationId = UUID.randomUUID().toString(); // ID unik untuk aplikasi          this.jobName = jobName;          this.applicantName = applicantName;          this.applicationDate = LocalDate.now().toString();          this.status = "Pending"; // Status awal      }      // Getter methods      public String getApplicationId() {          return applicationId;      }      public String getJobName() {          return jobName;      }      public String getApplicantName() {          return applicantName;      }      public String getApplicationDate() {          return applicationDate;      }      public String getStatus() {          return status;      }      // Metode untuk mengubah status aplikasi      public void updateStatus(String newStatus) {          this.status = newStatus;      }  } |

1. Class *Experience*

*Class Experience* ini berperan untuk menyimpan informasi tentang pengalaman kerja *JobSeeker*. Dimana class ini dirancang untuk menyimpan data berupa startDate, endDate, institution, dan description. *Class* ini sangat penting karena dapat digunakan oleh *Recruiter* untuk mencari kandidat atau pelamar dengan pengalaman tertentu.

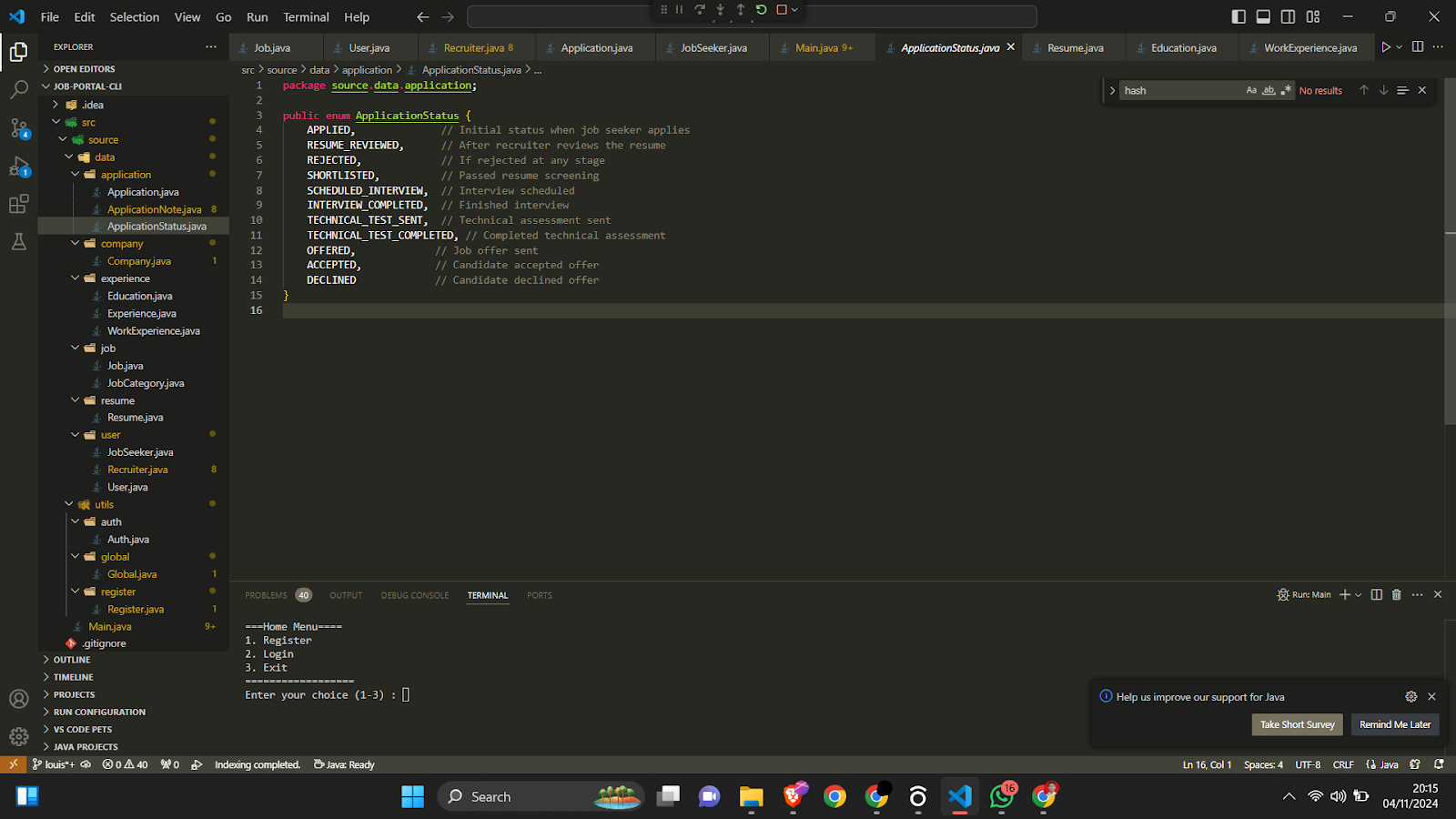
|  |
| --- |
| public class Experience {      private String startDate;      private String endDate;      private String institution;      private String description;      public Experience(String startDate, String endDate, String institution, String description) {          this.startDate = startDate;          this.endDate = endDate;          this.institution = institution;          this.description = description;      }      // Getter methods      public String getStartDate() {          return startDate;      }      public String getEndDate() {          return endDate;      }      public String getInstitution() {          return institution;      }      public String getDescription() {          return description;      }  } |

## 3.2 Pengimplementasian OOP

1. *Class* dan *Object*

        OOP tentunya tidak akan lepas dari konsep *class* dan *object*. Program kami telah mengimplementasikan delapan *class* yang dimana beberapa *class* tersebut dipecah lagi menjadi beberapa *class* kecil untuk memudahkan proses pengembangan aplikasi.

Untuk pengimplementasian konsep *object*, ketika program dijalankan, secara sadar maupun tidak sadar *object* akan terbentuk dengan sendirinya. Sebagai contoh ketika kita melakukan *register* sebagai *JobSeeker*, maka secara tidak langsung kita sudah membuat *object.*



*Gambar 3.2.1 - Penggunaan class pada program Job Portal*

1. *Inheritance*

*Inheritance* merupakan konsep penting dalam OOP dimana suatu *class* dapat mewarisi sifat dan perilaku pada class lain. Pada aplikasi *job portal* yang kami buat, terdapat *class user* yang digunakan sebagai *super class*. *Class* ini memiliki nama, email, nomor telepon, dan alamat. *User* merupakan *abstract class* yang tidak dapat dibuat langsung, namun digunakan sebagai dasar untuk class lain. Contoh *subclass* yang mewarisi *class user* adalah *class JobSeeker* dan *Recruiter*. *Class* tersebut secara otomatis memiliki semua atribut yang dimiliki *class user*.

|  |
| --- |
| public abstract class User {      private String name;      private String email;      private String phone;      private String address;      public User(String name, String email, String phone, String address) {          this.name = name;          this.email = email;          this.phone = phone;          this.address = address;      }      // Metode getter      public String getName() { return name; }      public String getEmail() { return email; }      public String getPhone() { return phone; }      public String getAddress() { return address; }  }  public class JobSeeker extends User {      public JobSeeker(String name, String email, String phone, String address) {          super(name, email, phone, address);      }  }  public class Recruiter extends User {      private String companyId;      public Recruiter(String name, String email, String phone, String address, String companyId) {          super(name, email, phone, address);          this.companyId = companyId;      }  } |

1. *Polymorphism*

*Polymorphism* adalah konsep dimana suatu *class* atau *object* dapat berperilaku berbeda tergantung pada *class* atau *object* yang memanggilnya. Untuk pengimplementasian *polymorphism,* Program akan memeriksa apakah *object* yang *login* merupakan *JobSeeker* atau  *Recruiter* dengan menggunakan *method instanceof*. Setelah diketahui *object* tersebut, maka kita bisa mengakses *method MenuJobSeeker()* atau *MenuRecruiter()*. *Polymorphism* membantu *developer* dalam menulis sebuah *code* secara rapi.

|  |
| --- |
| User<?> user = hashUsers.get(emailLogin);  if (user instanceof JobSeeker) {      MenuJobSeeker((JobSeeker) user);  } else if (user instanceof Recruiter) {      MenuRecruiter((Recruiter) user);  } |

1. *Abstraction*

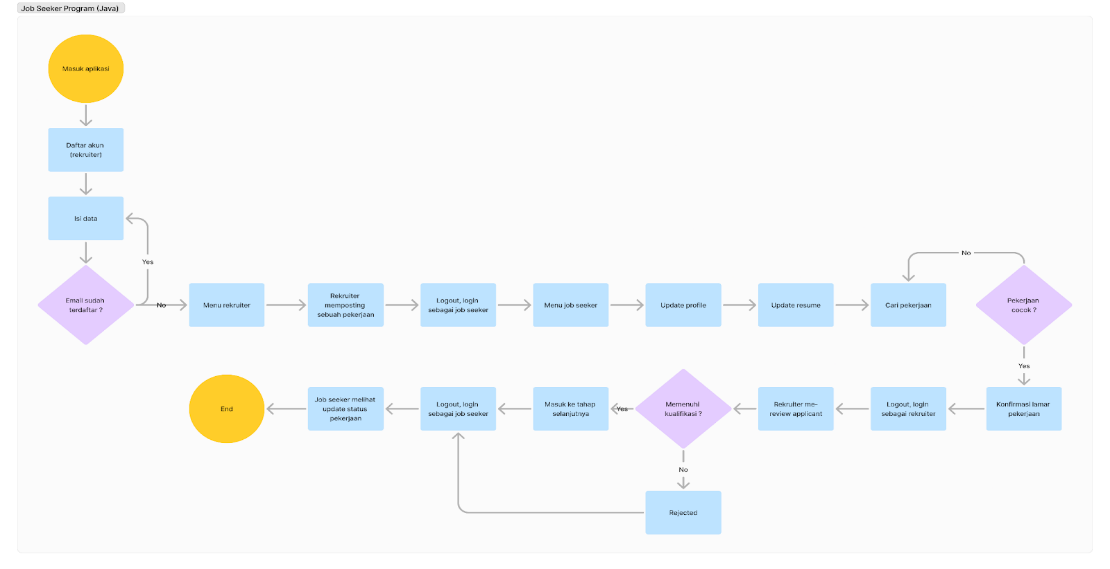
*Abstraction* merupakan salah satu konsep yang dapat menghilangkan detail yang rumit dari implementasi yang penting dari *class* atau *object*. *Abstraction* digunakan untuk sebagai *blueprint* dari suatu *class*, sehingga suatu *class* tinggal mengimplementasikan *method* bergantung pada kebutuhan suatu *class*. Di dalam kasus ini, *class user* dapat mendefinisikan metode abstrak di displayProfile().  *subclass JobSeeker* dan *Recruiter* dapat menerapkan sendiri sedangkan *class user* tidak diterapkan.

|  |
| --- |
| public abstract class User {      // kode yang ada      public abstract void displayProfile();  }  public class JobSeeker extends User {      // kode yang ada      @Override      public void displayProfile() {          System.out.println("Profil Job Seeker: " + getName());      }  }  public class Recruiter extends User {      // kode yang ada      @Override      public void displayProfile() {          System.out.println("Profil Recruiter: " + getName());      }  } |

1. *Encapsulation* (tambahan)

*Encapsulation* merupakan salah satu konsep yang dapat menghilangkan detail yang rumit dari implementasi yang penting dari *class* atau *object*. *Abstraction* digunakan untuk sebagai *blueprint* dari suatu *class*, sehingga suatu *class* tinggal mengimplementasikan *method* bergantung pada kebutuhan suatu *class*. Di dalam kasus ini, *class user* dapat mendefinisikan metode abstrak di displayProfile().  *subclass JobSeeker* dan *Recruiter* dapat menerapkan sendiri sedangkan *class user* tidak diterapkan.

## 3.3 Flowchart

Berikut adalah *flowchart* / alur bisnis dari program yang telah kami buat sebelumnya :

*Gambar 3.2.1 - Flowchart program*

## 3.4 Analisis SWOT

Berikut adalah analisis SWOT untuk aplikasi *Job Portal* yang sudah kami buat

* + - 1. **Strength**

1. *Job Portal* yang kami buat tidak memiliki *user interface* yang rumit, sehingga memudahkan pengguna baru untuk melakukan *register* atau *login*.
2. Dengan pemisahan *class* antara *jobseeker* dan *recruiter*, pengguna memiliki pengalaman.
3. Dengan banyaknya *class* yang digunakan serta berkembangnya *class* yang digunakan, maka program kami dapat dikembangkan lebih baik lagi.
   * + 1. **Weakness**
       2. *Job Portal* yang kami masih memiliki fitur dasar tanpa adanya fitur tambahan, seperti manajemen profil pengguna yang lebih *advanced* sehingga program dapat dikembangkan lagi
       3. Penggunaan *hashmap* sederhana sehingga pengguna dapat dengan mudah mengakses suatu akun hanya dengan mengetahui *email* seseorang.
       4. **Opportunity**
       5. *Job Portal* dapat dikembangkan lagi seperti manajemen profil pengguna, notif pemberitahuan, fitur *chat* antara *JobSeeker* dan *Recruiter.*
       6. Dengan integrasi teknologi baru, aplikasi *job portal* dapat dihubungkan dengan *database* sehingga aplikasi *job portal* dapat bekerja dengan baik.
       7. *Job Portal* ini dapat digunakan sebagai *project portofolio.*
       8. *Project job portal* adalah *project* yang jarang dipilih untuk tugas akhir.
       9. **Threat**
       10. *Job Portal* menghadapi persaingan ketat dengan *platform* yang lebih maju dan lebih terkenal seperti LinkedIn atau Indeed*.*
       11. Ekspetasi pengguna yang terbiasa menggunakkan *platform* lain sehingga cukup sulit untuk memenuhi ekspetasi tersebut.