BAB II

LANDASAN TEORI

2.1. Rekayasa Piranti Lunak

Menurut Pressman (2010, hal 14) Rekayasa Piranti Lunak adalah kumpulan dari aktivitas, aksi dan tugas yang dilakukan ketika sebuat produk akan dibuat. Sebuah aktivitas berusaha untuk mencapai objektif yang luas dan diterapkan tanpa memperhatikan domain aplikasi, ukuran dari proyek, kompleksitas dari pekerjaan atau tingkat ketelitian dimana rekayasa piranti lunak akan diterapkan.

Dalam konteks rekayasa piranti lunak, proses bukanlah ketentuan yang kaku untuk bagaimana membuat piranti lunak untuk komputer. Melainkan adalah pendekatan yang bisa beradaptasi yang memungkinkan orang – orang untuk memilih kumpulan pekerjaan yang paling tepat untuk masing – masing orang. Maksudnya adalah selalu memberikan piranti lunak dengan tepat waktu dan kualitas yang mumpuni untuk memenuhi kebutuhan dari semua pihak yang membiayai pembuatan piranti lunak tersebut dan untuk semua pihak yang akan menggunakannya.

Sebuah kerangka kerja untuk proses dibuat sebagai dasar untuk sebuah proses rekayasa piranti lunak yang lengkap dengan mengidentifikasi beberapa aktivitas kerangka kerja tersebut yang dapat diterapkan untuk semua proyek piranti lunak tanpa memperhatikan ukuran dan kompleksitasnya. Kerangka kerja yang umum pada rekayasa piranti lunak meliputi 5 aktivitas :

1. Komunikasi. Sebelum pekerjaan teknis dimulai, sangat penting untuk berkomunikasi dan berkolaborasi dengan pengguna. Maksudnya adalah untuk mengerti objektif dari pihak – pihak yang ada di perusahaan tentang proyek dan mengumpulkan kebutuhan yang dapat membantu untuk menetapkan fitur dan fungsi dari piranti lunak yang akan dibuat.
2. Perencanaan. Setiap perjalanan yang rumit dapat disederhanakan jika ada peta. Sebuah proyek piranti lunak adalah perjalanan yang rumit dan aktivitas perencanaan membuat "peta" yang membantu tim untuk dapat melalui perjalanan itu. Peta yang akan disebut rencana proyek piranti lunak menjelaskan tentang pekerjaan rekayasa piranti lunak dengan menggambarkan tugas – tugas yang akan dilakukan, resiko yang akan dihadapi, sumber daya yang diperlukan, produk yang akan diproduksi dan penjadwalan kerja.
3. Permodelan. Permodelan adalah sketsadari sesuatu sehingga tim dapat mengerti gambaran besar dari proyek, bagaimana perkiraannya, bagaimana bagian – bagian yang ada dapat terhubung dan banyak karakteristik yang lainnya. Jika diperlukan, rincian dari sketsa dapat dibuat lebih besar lagi untuk pemahaman yang lebih baik dan bagaimana cara untuk menyelesaikannya. Praktisi piranti lunak juga membuat model untuk lebih mengerti kebutuhan dari piranti lunak yang akan dibuat dan rancangan yang akan memenuhi kebutuhan – kebutuhan itu.
4. Konstruksi. Aktivitas ini menggabungkan pembuatan kode program dan pengujian yang diperlukan untuk mengevaluasi kesalahan yang ada pada kode.

5. Penerapan. Piranti lunak yang sudah diselesaikan atau masih sebagian diberikan kepada pemakai yang mengevaluasi produk yang diberikan dan pemakai memberikan timbal balik atas produk yang telah dievaluasi.

   (Pressman, 2010, hal 15)

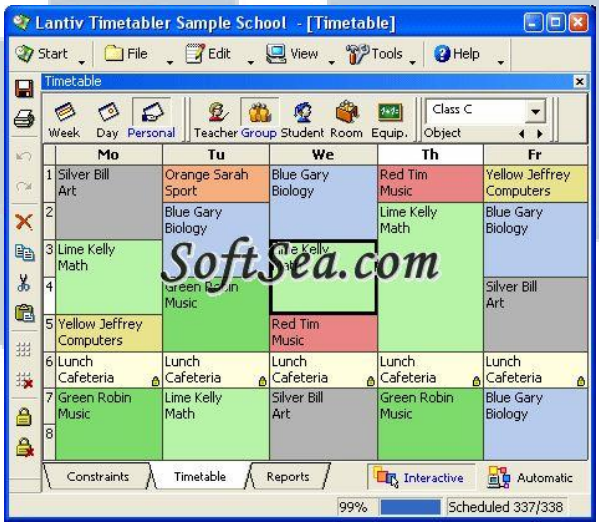
2.2. Piranti Lunak Penjadwalan

Piranti Lunak Penjadwalan adalah piranti lunak khusus yang dirancang untuk menyelesaikan masalah pembuatan jadwal pada suatu institusi atau suatu sistem yang berjalan dengan jumlah sumber daya yang terbatas. Ada beberapa piranti lunak yang sudah pernah dibuat untuk penjadwalan sekolah diantaranya adalah *Lantiv Time, ASC Time Table* dan *FET*.

2.2.1. Lantiv Time

Lantiv Time adalah piranti lunak yang diproduksi oleh Lantiv International (2007, http://www.lantiv.com/) pada tahun 2007 untuk membuat jadwal dan juga pembentuk tabel waktunya. Program ini memungkinkan penjadwalan secara interaktif dan otomatis. Program ini mendukung sistem penjadwalan menggunakan blok dan juga memungkinkan untuk menjadwalkan ruangan serta perlengkapan yang diperlukan. Program ini mampu membuat jadwal untuk sekolah yang memiliki fitur untuk menyusun jadwal siswa, kelas, guru, ruangan yang digunakan membagi siswa ke dalam beberapa grup, menyusun jadwal kelas yang simultan dan juga melihat jadwal dari masing – masing siswa. Program ini juga mampu untuk menjadwal kegiatan di rumah sakit seperti menyusun jadwal dokter dan suster ke dalam beberapa *shift* dan

penggunaan kamar yang ada. Di samping itu juga terdapat fitur untuk memasukkan jam kerja tertentu, menetapkan hari libur, menghasilkan laporan yang rinci, mencetak jadwal dokter dan suster serta fitur untuk melihat jadwal bulanan, mingguan dan harian.



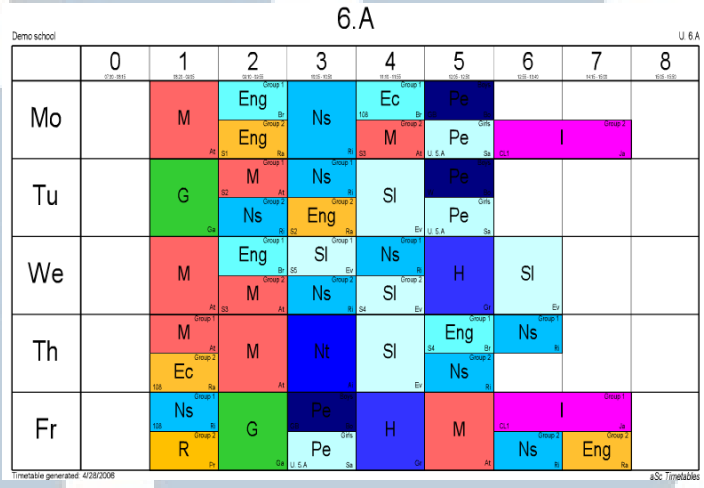
Gambar 2.1 Piranti Lunak Lantiv Time

2.2.2. ASC Timetable

ASC Timetable adalah piranti lunak yang dibuat oleh Applied Software Consultants (1997, http://www.asctimetables.com/timetables_en.html) pada tahun 1997 yang berlokasi di Slovakia. ASC Timetable digunakan untuk menyusun jadwal di sekolah dasar dan sekolah lanjutan tingkat pertama. Beberapa fitur yang dimiliki adalah :

1. Entri data yang mudah
2. Membuat jadwal secara otomatis

3. Pemeriksaan hasil dan pencetakan jadwal.



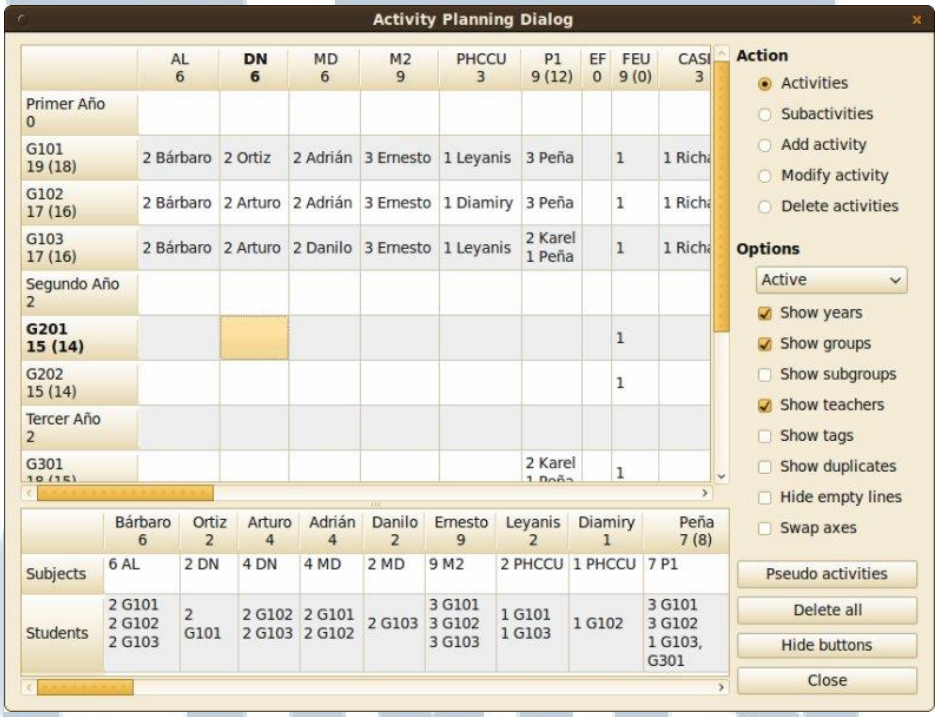
Gambar 2.2 Piranti Lunak ASC Timetable

2.2.3. FET

FET adalah piranti lunak sumber terbuka yang dibuat oleh Liviu Lalescu (2009, http://www.lalescu.ro/liviu/fet/) pada tahun 2009 untuk membuat jadwal secara otomatis di sekolah dan juga universitas. FET memiliki beberapa fitur yaitu :

1. Dilokalisasi ke beberapa bahasa yaitu Inggris, Arab, Belanda, Jerman, Yunani, Spanyol dan Indonesia.
2. Dapat membuat jadwal secara otomatis, setengah otomatis dan juga manual.
3. Dapat digunakan pada beberapa macam *platform* seperti Microsoft Windows, Mac atau Linux dan semua sistem yang mendukung Qt
4. Dapat mengimpor dan mengekspor data dari format CSV.

5. Hasil pembuatan jadwal dapat diekspor menjadi HTML, XML dan CSV.
6. Struktur penyusunan siswa yang sangat fleksibel dari pengumpulan grup berdasarkan tahun angkatan, grup dan subgrup.
7. *Constraint* yang ditentukan memiliki bobot persentase yang dapat diatur dari 0 % hingga 100 %.



Gambar 2.3 Piranti Lunak FET

2.3. Algoritma *Greedy*

Algoritma *Greedy* adalah algoritma yang selalu mengambil pilihan yang paling optimal dari situasi dan kondisi tertentu. Oleh karena itu, pilihan – pilihan yang ditentukan akan menjadi optimum lokal yang dengan harapan akan menuju solusi

yang optimum secara global. Algoritma *Greedy* tidak selalu berakhir dengan solusi yang optimal, tetapi banyak permasalahan yang solusi akhirnya optimal.

(Cormen, et al. , 2001, hal 370)

2.3.1. Elemen Algoritma *Greedy*

Menurut Cormen, dkk (2001, hal 379) Algoritma *Greedy* mendapatkan solusi optimal untuk masalah dengan membuat serangkaian pilihan. Pada setiap titik pemilihan di dalam algoritma, pilihan yang paling optimal dari situasi dan kondisi tertentu yang dipilih. Strategi heuristis ini tidak selalu menghasilkan solusi yang optimal. Dalam mengembangkan algoritma *greedy* diperlukan beberapa langkah yaitu :

1. Menentukan substruktur optimal dari masalah yang mana pilihan – pilihan telah dibuat dan menyisakan 1 submasalah untuk diselesaikan.
2. Membuktikan bahwa selalu ada solusi optimal dari masalah utama yang membuat pilihan *greedy* selalu aman.
3. Memperlihatkan bahwa dengan menggabungkan solusi optimal dari submasalah dapat menghasilkan solusi optimal dari masalah secara keseluruhan.

2.3.2. Properti Pemilihan *Greedy*

Cormen, dkk (2001, hal 380) menjelaskan bahwa properti pemilihan *greedy* yaitu solusi optimal secara global dapat dicapai dengan membuat pilihan yang optimal secara lokal. Dengan kata lain, saat mempertimbangkan pilihan mana yang akan

dibuat, pilihan yang dibuat adalah yang nampak paling baik dalam masalah saat itu, tanpa mempertimbangkan hasil dari submasalah. Ini yang membuat perbedaan dengan *dynamic programming.* Dalam *dynamic programming*, pilihan dibuat setiap langkah namun biasanya pilihan bergantung pada solusi ke submasalah. Sebagai konsekuensi, penyelesaian dari *dynamic programming* adalah *bottom – up,* bergerak dari submasalah yang kecil menuju submasalah yang besar. Pada algoritma *greedy*, pilihan yang dibuat adalah yang nampaknya terbaik saat itu dan menyelesaikan submasalah yang timbul setelah pilihan dibuat. Pilihan yang dibuat oleh algoritma *greedy*, mungkin tergantung pada pilihan – pilihan yang sudah dibuat, tetapi tidak dapat tergantung pada apapun pilihan berikutnya atau kepada solusi dari submasalah. Oleh karena itu, tidak seperti *dynamic programming* yang menyelesaikan masalah *bottom – up,* algoritma *greedy* biasanya bergerak secara *top – down,* membuat pilihan *greedy* selanjutnya, mengurangi masalah yang diberikan menjadi lebih kecil.

Properti pemilihan *greedy* sering memberikan efisiensi dalam membuat pilihan pada submasalah. Sangat sering pada kasus yang dengan pemrosesan awal dari masukan atau menggunakan struktur data yang tepat, pilihan *greedy* dapat dibuat dengan cepat dan menghasilkan algoritma yang efisien.

Sebagai contoh perbandingan penyelesaian masalah antara *dynamic programming* dan algoritma *greedy* untuk kasus pengembalian uang. Dimisalkan di suatu negara hanya terdapat 3 jenis nominal uang yaitu 1, 5 dan 7. Untuk kasus pengembalian uang sebesar 21 langkah yang dilakukan oleh algoritma *greedy* adalah memilih 3 keping nominal 7 secara langsung karena algoritma *greedy* memilih secara *top – down* yang

dalam hal ini adalah memilih nominal uang terbesar terlebih dahulu. Sedangkan yang dilakukan oleh metode *dynamic programming* adalah secara *bottom – up* yang dimana memilih nominal 1 dan disimpan sementara banyaknya keping adalah 21 keping nominal 1, kemudian menghitung jika menggunakan nominal 5 dan disimpan banyaknya keping adalah 5 yang terdiri dari 4 keping nominal 5 dan 1 keping nominal 1, kemudian menyimpan nilai akhir sebanyak 3 keping nominal 7.

2.3.3. Substruktur Optimal

Cormen, dkk (2001, hal 381) juga menjelaskan bahwa sebuah masalah menghasilkan substruktur optimal jika diperoleh hasil optimal pada setiap submasalah. Biasanya lebih banyak menggunakan pendekatan langsung mengenai substruktur optimal saat diterapkan pada algoritma *greedy*. Oleh karena itu yang harus benar – benar dilakukan adalah menyatakan bahwa solusi optimal dari submasalah, dikombinasikan dengan pilihan *greedy* yang telah dibuat, menghasilkan solusi optimal dari masalah yang ada. Skema ini secara implisit menggunakan induksi pada submasalah untuk membuktikan bahwa membuat pilihan *greedy* di setiap langkah menghasilkan solusi optimal.

2.4. Transform and Conquer

Di dalam bukunya, Levitin (2006, hal 197) menjelaskan bahwa *Transform and Conquer* adalah kelompok teknik penyelesaian masalah dengan transformasi. Beberapa macam transformasi adalah :

1. Membuat instansi atau data yang diproses menjadi lebih sederhana atau mudah diproses yang disebut juga penyederhanaan instansi.
2. Transformasi menjadi representasi yang berbeda dari instansi yang sama.
3. Transformasi menjadi instansi dari masalah yang berbeda yang algoritma untuk menyelesaikan masalah tersebut sudah ada.

Ada beberapa metode atau algoritma yang mengadaptasi konsep *Transform and Conquer*, salah satunya adalah *Presorting. Presorting* merupakan ide lama dalam disiplin ilmu komputer. Faktanya, ketertarikan dalam algoritma *sorting* membuat perbedaan yang signifikan menuju kepada fakta yang banyak soal mengenai koleksi atau *list* lebih mudah diselesaikan jika koleksi tersebut sudah diurutkan. Sudah jelas bahwa efisiensi waktu dari algoritma yang melibatkan pengurutan akan bergantung pada algoritma mengurutkan yang digunakan. Contoh soal yang cukup mudah diselesaikan dengan *presorting* adalah mencari nilai tengah dari kumpulan angka. Jika kumpulan tersebut diurutkan dahulu, maka solusinya adalah datum yang terletak di tengah. (Levitin, 2006, hal 198)

2.5. Database dan DBMS

Menurut Connolly dan Begg (2009, hal 15), *Database* adalah kumpulan dari data yang secara logika memiliki relasi yang digunakan secara berbagi dan dirancang untuk memenuhi kebutuhan informasi dari suatu organisasi. *Database* adalah sesuatu, kemungkinan tempat penyimpanan yang besar dar data yang dapat digunakan secara

simultan oleh banyak departemen dan pengguna. Daripada menggunakan berkas di komputer masing – masing dengan data yang redundan, semua data diintegrasikan dengan jumlah duplikasi yang minimum. Oleh karena itu *database* tidak lagi dimiliki oleh satu departemen saja, melainkan merupakan sumber daya berbagi dari suatu perusahaan.

Ekspresi lain dalam definisi dari *database* yang harus dijelaskan adalah "berelasi secara logika". Saat menganalisis kebutuhan informasi dari suatu organisasi yang akan direpresentasikan dalam *database,* perancang mencoba mengidentifikasi entitas, atribut dan relasi. Sebuah entitas adalah objek yang berbeda – beda dalam organisasi yang akan direpresentasikan di dalam *database.* Sebuah atribut adalah properti yang menggambarkan beberapa aspek dari objek yang akan disimpan, dan relasi adalah asosiasi antara entitas.

Sedangkan DBMS atau *Database Management System* adalah sistem piranti lunak yang memungkinkan pengguna untuk mendefinisikan, membuat, mengelola dan mengendalikan akses ke *database*. Biasanya DBMS menyediakan fasilitas berikut :

1. Memungkinkan pengguna untuk mendefinisikan *database,* biasanya melalui *Data Definition Language* atau DDL. DDL memperbolehkan pengguna untuk menetapkan tipe data, struktur dan batasan pada data yang disimpan dalam *database.* Contoh dari DDL adalah CREATE, ALTER, RENAME, DROP dan TRUNCATE.
2. Memperbolehkan pengguna untuk memasukkan, membarui, menghapus dan mengambil data dari *database,* biasanya melalui *Data Manipulation*

Language atau DML. Contoh dari DML adalah SELECT, INSERT, UPDATE dan DELETE. Memiliki tempat penyimpanan terpusat untuk semua data dan deskripsi data menyebabkan DML untuk menyediakan fasilitas *inquiry* yang disebut bahasa kueri. Bahasa kueri yang paling umum adalah *Structured Query Language* atau SQL.

3. Menyediakan akses terkendali ke database. Misalnya menyediakan fasilitas sistem keamanan, sistem terpadu, sistem kontrol berbagi, sistem kontrol pemulihan dan katalog data.(Connolly dan Begg, 2009, hal 15)

2.6. Perancangan Interface

2.6.1. Pedoman Perancangan *User Interface*

Dari awalnya era komputerisasi, perancang *interface* telah menuliskan pedoman untuk mencatat pengetahuannya dan mencoba untuk memandu dalam upaya untuk perancang – perancang selanjutnya. Pedoman dari *Apple* dan *Microsoft* yang mempengaruhi perancang *desktop-interface,* telah diikuti oleh banyak dokumen yang berisi tentang pedoman perancangan *user interface*.

Namun, banyak kritikus yang melakukan protes karena pedoman dapat menjadi terlalu spesifik, tidak lengkap, sulit untuk diterapkan dan kadang – kadang salah. Argumen pendukung yang dibangun berdasarkan pengalaman dari perancang ahli berkontribusi untuk peningkatan kualitas *interface*. (Shneiderman, et al. , 2009, hal 57)

A. Organisasi tampilan

Perancangan tampilan adalah topik besar dengan banyak kasus. Smith dan Mosier pada tahun 1986 (Shneiderman, et al. , 2009, hal 58) memberikan lima sasaran sebagai bagian dari pedoman mereka tentang tampilan data :

1. Konsistensi tampilan data.
2. Perpaduan informasi yang efisien oleh pengguna.
3. Minimalisasi beban ingatan pengguna.
4. Kompatibilitas tampilan data dengan pemasukan data.
5. Fleksibilitas kendali oleh pengguna dari tampilan data.

B. Mendapatkan perhatian pengguna

Shneiderman (2009, hal 60) mengungkapkan bahwa informasi subtansial akan ditampilkan kepada pengguna untuk pekerjaan yang biasa dilakukannya, kondisi pengecualian atau informasi yang bergantung pada waktu harus ditampilkan untuk mendapatkan perhatian penggguna :

1. Intensitas. Menggunakan hanya 2 tingkatan saja.
2. Penandaan. Menggarisbawahi objek, menunjuk objek dengan tanda panah atau menggunakan indikator seperti *asterisk, dash,* bulatan dan sebagainya.
3. Ukuran. Gunakan sampai dengan 4 ukuran, dengan ukuran yang besar lebih banyak menarik perhatian.
4. Pemilihan *font.* Menggunakan sampai dengan 3 jenis *font.*
5. *Inverse video.* Menggunakan pewarnaan yang kontras satu dengan yang lain.
6. Kelap – kelip. Menampilkan teks, gambar atau warna – warna yang kelap – kelip dengan hati – hati dan dalam area tertentu.
7. Warna. Menggunakan sampai dengan 4 warna standar, dengan tambahan warna yang dipersiapkan untuk pemakaian tambahan.
8. Suara. Menggunakan bunyi yang lembut untuk timbal balik yang positif dan suara yang parau untuk kondisi – kondisi kesalahan tertentu yang cukup fatal.

C. Memfasilitasi entri data

Tugas entri data dapat menyita waktu pengguna cukup lama dan dapat menjadi sumber frustasi dan memungkinkan terjadinya kesalahan yang berbahaya. Smith dan

Mosier pada tahun 1986 (Shneiderman, et al. , 2009, hal 61) memberikan 5 objektif sebagai bagian dari pedoman mereka tentang entri data:

1. Konsistensi transaksi pemasukan data.
2. Minimalisasi aksi pemasukan dari pengguna.
3. Minimalisasi beban ingatan pengguna.
4. Kompatibilitas input data dengan tampilan data.
5. Fleksibilitas kendali pengguna.

2.6.2. Prinsip Perancangan *User Interface*

Shneiderman (2009, hal 62) juga berpendapat, disaat pedoman difokuskan semakin sempit, prinsip – prinsip cenderung untuk lebih mendasar, diterapkan secara luas dan terus – menerus. Contohnya adalah menetapkan tingkat kemampuan pengguna, Mempelajari tentang pengguna adalah ide yang sederhana tetapi sulit dan sayangnya biasanya tujuannya diremehkan. Tidak seorangpun akan membantah tentang prinsip ini, namun banyak perancang dengan mudahnya mengasumsikan bahwa mereka mengerti para pengguna dan juga tugas dari pengguna tersebut. Shneiderman (2009, hal 63 – 64) menyarankan pemisahan tingkat kemampuan pengguna dibagi berdasarkan perbedaan tujuan akhir perancangan yaitu:

1. *Novice* atau pengguna biasa.
2. *Knowledge intermittent users* atau pengguna yang memiliki cukup pengetahuan.
3. *Expert frequent users* atau pengguna ahli.

2.7. Bahasa Pemrograman Java

Menurut Gosling, dkk (2005, hal 1) yang adalah pengembang dari Bahasa Pemrograman *Java, Java* adalah bahasa pemrograman yang digunakan untuk kebutuhan umum seperti sistem pakar, aplikasi basis data dan yang lainnya. Bahasa Java bersifat *concurrent*, berbasis kelas dan berorientasi objek. *Java* dirancang cukup sederhana yang banyak pemrogram dapat dengan cepat terbiasa menggunakan bahasa tersebut. Bahasa Pemrograman *Java* berelasi dengan C dan C++ tetapi diorganisasikan cukup berbeda, dengan beberapa aspek dari C dan C++ yang dihilangkan dan mengikutsertakan beberapa ide dari bahasa pemrograman lain.

Bahasa Pemrograman *Java* adalah bahasa yang relatif tingkat tinggi dengan perincian dari representasi bahasa mesin tidak tersedia melalui *Java*. Itu termasuk *Automatic Storage Management* yang biasanya menggunakan *garbage collector* untuk menghindari masalah keamanan dari pembersihan memori secara eksplisit yang dilakukan pada bahasa C dan C++. Implementasi pengumpulan sampah dengan kinerja tinggi dapat dibatasi untuk mendukung pemrograman sistem dan aplikasi *real time.*