Software este un termen general pentru diferite tipuri de programe folosite la operarea calculatoarelor si a dispozitivelor inrdudite. Astfel software-ul poate fi vazut ca partea care este variabila a unui calculator si hardware-ul partea invariabila .

Software-ul este des divizat in aplicati software si software de sistem. Aplicatile software sunt programe care realizeaza o sarcina in care utilizatorul este direct interesat de rezultat. Software de sistem se refera la software-ul care include sistemul de operare si orice program care il ajuta .

Uneori abreviat ca SW sau S / W, software-ul este o colecție de instrucțiuni care permit utilizatorului să interacționeze cu hardware-ul său, un calculator , sau să îndeplinească sarcini. Fără software, majoritatea calculatoarelor ar fi inutile.

Software-ul este creat de un programator care scrie instructiuni folosind un un limbaj de programare pentru a ii spune software-ului cum sa functioneze si ce sa faca. Odata ce programul este coplet, el este compilat intr-un limbaj ce calculatorul il poate intelege.

Deseori se face confuzia intre termenul software si program. Desigur termenii de software și de program sunt utilizați în mod interschimbabil, deoarece deseori se referă la același lucru în utilizarea zilnică . Cu toate ca sunt termini foarte apropiati find apropape sinonime, exist amici diferente de care trebuie tinut cont. Cuvantul Software este un termen foarte larg care contine programele, datele și alte fișiere conexe care sunt folosite pentru a îndeplini anumite sarcini într-un calculator sau pe orice alt dispozitiv care efectuează o activitate de calcul. În acest sens, putem spune că chiar un program este și un software. Dar, în sensul mai larg al cuvântului, un program este orice set de instrucțiuni care sunt executate de o mașină. De exemplu, să presupunem că aveți un software care înregistrează nume și adrese într-o bază de date. Programul și baza de date sunt părți ale software-ului, dar baza de date nu este parte din program. Este pur și simplu un accesoriu al programului care îl face mai util.

Din punct de vedere chronologic programele au existat inainte de software. Prima dată când un calculator a ținut o bucată de software într-o memorie electronică și l-a executat cu succes, a fost 11am, 21 iunie 1948, la Universitatea din Manchester, pe mașina experimentală Small Scale, cunoscută și sub numele de "Baby ". Ea a fost scrisă de Tom Kilburn și a calculat cel mai înalt factor al întregului număr 2 ^ 18 = 262.144. Google a dat un omagiu lui Manchester Baby, sărbătorind ca "nașterea software-ului".

Java este un limbaj de programare utilizat pe scară largă, conceput în mod expres pentru utilizarea în mediul distribuit al internetului. Este cel mai popular limbaj de programare pentru aplicațiile Android și se numără printre cele mai favorizate pentru dezvoltarea dispozitivelor și a internet of things.

Java a fost conceput pentru a fi similar limbajului C ++, dar este mai simplu de utilizat decât C ++ și impune un model de programare orientat pe obiecte. Java poate fi folosit pentru a crea aplicații complete care pot fi rulate pe un singur computer sau pot fi distribuite între servere și clienți dintr-o rețea. Acesta poate fi, de asemenea, utilizat pentru a construi un modul de aplicație mic sau un applet pentru utilizare ca parte a unei pagini Web.

Este dificil să furnizăm un singur motiv pentru care limbajul Java a devenit atât de popular. Cu toate acestea, caracteristicile majore ale limbii au jucat un rol în succesul său, incluzând următoarele componente:

* Programele create în Java oferă portabilitate într-o rețea. Codul sursă este compilat în bytecode, care poate fi rulat oriunde într-o rețea pe un server sau client care are o mașină virtuală Java (JVM). JVM interpretează codul bytecode în cod care va rula pe hardware-ul calculatorului. În schimb, majoritatea limbajelor de programare, compilează codul într-un fișier binar. Fișierele binare sunt specifice platformei, deci un program scris pentru o mașină Windows bazată pe Windows nu poate rula pe un Mac. JVM include un compilator opțional Just In Time(JIT), care compilează dinamic codul bytecode în cod executabil ca alternativă la interpretarea simultană a unei instrucțiuni bytecode. În multe cazuri, compilația dinamică JIT este mai rapidă decât interpretarea mașinii virtuale.
* Codul este robust. Spre deosebire de programele scrise în C ++ și alte limbaje, obiectele Java nu conțin referințe la date externe sau alte obiecte cunoscute. Acest lucru asigură faptul că o instrucțiune nu poate conține adresa de stocare a datelor într-o altă aplicație sau în sistemul de operare în sine, oricare ar determina încetarea sau prăbușirea programului și chiar a sistemului de operare. JVM face o serie de verificări pe fiecare obiect pentru a asigura integritatea.
* Java este un limbaj orientat pe obiecte. Un obiect poate profita de faptul că face parte dintr-o clasă de obiecte și mosteneste cod care este comun clasei. Obiectele sunt considerate "substantive" pe care un utilizator le-ar putea referi mai usor decât "verbele" din programarea procedurala. O metodă poate fi considerată ca fiind una dintre capabilitățile sau comportamentele obiectului.. Abilitatea de a dezvolta cu un limbaj creat de la început pe orientarea obiectului ca scop explicit a făcut Java o platformă interesantă.
* Dezvoltatorii pot învăța Java rapid. Cu sintaxa similară cu C ++, Java este relativ ușor de învățat, mai ales pentru cei cu un background în C. Deasemena Java dispune de o cominitate bogata, facilitand un numar mare de materiale destinate invatatri limbajului si avand o serie de solutii pentru probleme des intalnite.

Există trei platforme-cheie pe care programatorii dezvoltă aplicații Java:

* Java SE. Aplicațiile simple, stand-alone, sunt dezvoltate folosind Java Standard Edition. In trecut cunoscut sub numele de J2SE, Java SE oferă toate API-urile necesare dezvoltării aplicațiilor desktop tradiționale.
* Java EE. Java Enterprise Edition, cunoscut anterior ca J2EE, oferă posibilitatea de a crea componente pe partea de server care pot răspunde la un ciclu de solicitare-răspuns. Acest aranjament permite crearea de programe Java care pot interacționa cu clienți bazați pe internet, inclusiv browsere web, și chiar servicii web bazate pe REST și SOAP.
* Java ME. Java oferă, de asemenea, o platformă ușoară pentru dezvoltarea mobilă cunoscută sub numele de Java Micro Edition, cunoscută anterior ca J2ME. Java ME sa dovedit a fi o platformă foarte populară pentru dezvoltarea dispozitivelor embedded, dar sa străduit să obțină suprematia în piata de dezvoltare a smartphone-urilor. În ceea ce privește dezvoltarea de smartphone-uri, Android a devenit platforma de dezvoltare mobilă.

Dupa cum am mentionat mai sus Java este un limbaj orientat pe obiect(OOP). Este vital sa se respecte niste principi pentru a screi un program de calitate din punt de vedere al programari orientate pe obiect. Aceste principi sunt :

* Încapsularea, este un mecanism care combină codul și datele pe care le manipulează, menținând integritatea acestora față de interferența cu lumea exterioară. Incapsularea mai este numită și realizarea de cutii negre, întrucât se ascunde functionalitatea proceselor. Cand codul și datele sunt încapsulate se creaza un obiect.
* Polimorfismul, este calitatea care permite unei interfețe să aibă acces la un grup generic de acțiuni. Termenul este derivat dintr-un cuvânt grecesc având semnificația “cu mai multe forme”. Să presupunem că avem o nevoie de o funcție care să returneze aria unei forme geometrice, care poate fi un triunghi, cerc sau trapez. Întrucât ariile celor trei forme se calculează diferit, routina trebuie să fie adaptată la datele pe care le primește încât să distingă despre ce fel de formă este vorba și să returneze rezultatul corect.
* Moștenirea, este procesul prin care un obiect poate dobândi caracteristicile altui obiect. Utilizând moștenirea, se poate crea o clasă generală, care definește caracteristici comune unei mulțimi și care poate fi moștenită de alte clase, mai specifice, fiecare adăugând numai caracteristicile care o identifică în mod unic. O clasă care este moștenită se numeste clasă de bază, iar clasa care moștenește anumite caracteristici se numește clasă derivată.
* Reutilizarea, atunci când este creată o clasă, aceasta poate fi utilizată pentru a crea o mulțime de obiecte. Prin utilizarea moștenirii și încapsulării clasa amintită poate fi reutilizată. Nu mai este nevoie să testăm codul respectiv ci doar să îl utlizam corect.

O mașină virtuală Java (JVM) este o mașină abstractă de calcul care permite unui computer să ruleze un program Java. Există trei noțiuni ale JVM: specificație, implementare și instanță. Specificația este un document care descrie în mod oficial ceea ce este necesar pentru implementarea JVM. Având o singură specificație, toate implementările sunt interoperabile. Implementarea JVM-ului este un program de calculator care îndeplinește cerințele specificației JVM. O instanță a unui JVM este o implementare care rulează într-un proces ce execută un program compilat în Java bytecode.

Java Runtime Environment (JRE) este un pachet software care conține tot ce este necesar pentru a rula un program Java. Acesta include implementarea unei mașini virtuale Java împreună cu o implementare a Bibliotecii de clase Java. Oracle Corporation, care deține marca comercială Java, distribuie un mediu Java Runtime cu mașina lor virtuală Java denumită HotSpot.

Kitul de dezvoltare Java (JDK) este o superset al unui JRE și conține instrumente pentru programatori Java. Kitul de dezvoltare Java este oferit gratuit fie direct de Oracle Corporation, fie prin proiectul open source OpenJDK, care este guvernat de Oracle.

Un integrated development environment (IDE) este o suită de software-uri care consolidează instrumentele de care dezvoltatorii au nevoie ca să scrie și să testeze programele. De obicei, un IDE conține un editor de cod, un compilator sau un interpretor și un program de depanare(debugger) pe care utilizatorul le accesează printr-o singură interfață grafică (GUI). Un IDE poate fi o aplicație independentă sau poate fi inclusă ca parte a uneia sau mai multor aplicații existente și compatibile.

Eclipse este un IDE utilizat în programare și este cel mai utilizat IDE de Java. Acesta conține un spațiu de lucru de bază și un sistem plug-in extensibil pentru personalizarea mediului. Eclipse este scrisă în principal în Java și utilizarea sa principală este pentru dezvoltarea de aplicații Java, dar poate fi folosit și pentru dezvoltarea de aplicații în alte limbaje de programare prin plug-in-uri.Acesta poate fi, de asemenea, utilizat pentru a dezvolta documente cu LaTeX .

Eclipse a luat nastere în anul 2001, când IBM a donat trei milioane de linii de cod din instrumentele sale Java. Scopul original al Eclipse-ului a fost de a crea și de a promova o comunitate open source IDE care să completeze comunitatea care înconjoară Apache. Zvonurile spun că un obiectiv secundar a fost "eclipsarea Microsoft Visual Studio", acesta este modul în care platforma și-a luat numele.

Principalul avantaj din punctul meu de vedere este documentatia bogata de care dispune acest IDE. Platforma Eclipse și alte plug-in-uri de la fundația Eclipse sunt lansate sub licența Eclipse Public License (EPL). EPL asigură că Eclipse este o solutie gratis. De asemenea, Eclipse permite modificarea și distribuirea. În fiecare an, începând cu anul 2006, fundația Eclipse lansează platforma Eclipse și o serie de alte plug-in-uri în luna iunie.

JavaFX este o platformă software pentru crearea și furnizarea de aplicații desktop, precum și aplicații web bogate (RIA) care pot rula pe o gamă largă de dispozitive. JavaFX este destinat să înlocuiască Swing ca bibliotecă grafică standard pentru Java SE, dar ambele vor fi incluse în viitorul previzibil . JavaFX are suport pentru desktop și browserele web pe Microsoft Windows, Linux și MacOS.

Înainte de versiunea 2.0 a JavaFX, dezvoltatorii au folosit o limbă statică, declarativă, numită JavaFX Script, pentru a construi aplicații JavaFX. Deoarece JavaFX Script este compilat în Java bytecode, programatorii pot folosi de asemenea cod Java. Aplicațiile JavaFX ar putea rula pe orice desktop care ruleaza Java SE, pe orice browser care poate rula Java EE, sau pe orice telefon mobil care poate rula Java ME.

JavaFX 2.0 și versiunile urmatoare sunt implementate ca o bibliotecă Java "nativă", iar aplicațiile care folosesc JavaFX sunt scrise în cod Java "nativ". Scriptul JavaFX a fost dezactivat de Oracle. JavaFX 2.x nu suportă sistemul de operare Solaris sau telefoanele mobile; Cu toate acestea, Oracle intenționează să integreze JavaFX în Java SE Embedded 8, iar Java FX pentru procesoarele ARM se află în faza de previzualizare a dezvoltari.

Pe desktop-uri, JavaFX suportă sistemele de operare Windows Vista, Windows 7, Windows 8, Windows 10, MacOS și Linux. Pe mobil, JavaFX Mobile 1.x este capabil să ruleze pe mai multe sisteme de operare mobile, inclusiv Symbian OS, Windows Mobile,etc.

JavaFX este proiectat pentru a oferi dezvoltatorilor Java o nouă platformă grafică ușoară și performantă. Intenția este ca noile aplicații să utilizeze JavaFX mai degrabă decât Swing pentru a construe o interfața grafică (GUI) a aplicației. Acest lucru nu înseamnă că Swing este depășit. Numărul mare de aplicații care au fost construite folosind Swing înseamnă că va fi încă parte din Java API mai ales că aceste aplicații pot încorpora funcționalitatea JavaFX deoarece cele două API-uri grafice se execută una lângă cealaltă.

Acest lucru însemna că dezvoltatorii Java nu au nevoie să învețe un nou limbaj grafic și pot confortabil sa creeze o apliecatie JavaFX folosind sintaxa obișnuită Java. API-ul JavaFX conține tot ce v-ați aștepta de la o platformă grafică - controale UI, animații, efecte etc.

Principala diferență pentru dezvoltatorii care trec de la Swing la JavaFX va fi obișnuita cu modul în care sunt prezentate componentele grafice și noua terminologie. O interfață de utilizator este încă construită utilizând o serie de straturi care sunt conținute într-un grafic de scenă. Graficul este afișat pe un container de nivel superior numit etapă.

Alte caracteristici importante odata JavaFX 2.0 sunt:

* Un nou motor grafic - Prism, o conductă hardware accelerată, este cuplată cu Glass, ca o fereastră nouă, pentru a produce grafică de înaltă calitate pentru aplicațiile JavaFX.
* Un nou limbaj de marcare declarativă denumită FXML. Este bazat pe XML și permite dezvoltatorilor să definească o interfață pentru o aplicație JavaFX.
* Un nou motor media pentru redarea conținutului multimedia web.
* Un plug-in de browser pentru încărcarea applet-urilor JavaFX utilizând Prism.
* Componenta web pentru încorporarea paginilor web într-o aplicație JavaFX.
* Un doclet pentru generarea documentației API JavaFX utilizând Javadoc

Există, de asemenea, exemple de aplicații Java care vin împreună cu SDK pentru a arăta dezvoltatorilor cum să construiască diferite tipuri de aplicații JavaFX. Pentru utilizatorii de Windows, JavaFX SDK face parte din Java SE JDK de la actualizarea Java 7. De asemenea, JavaFX rulează acum Java SE JRE.

FXML este un limbaj bazat pe XML care oferă structura pentru construirea unei interfețe de utilizator separate de logica aplicației. Această separare a logicii de prezentare și aplicație este atractivă pentru dezvoltatorii pentru că pot dezvolta o interfață de utilizator care folosește componentele Java fără a stăpâni codul pentru preluarea și completarea datelor.

FXML are o structură predefinită de bază. Ce puteți să exprimați în FXML și cum se aplică pentru construirea unei scene depinde de API-ul obiectelor pe care le construiți. Deoarece FXML se aplica direct în Java, puteți utiliza documentația pentru a înțelege ce elemente și atribute sunt permise. În general, majoritatea calaselor JavaFX pot fi folosite ca elemente, iar cele mai multe proprietăți pot fi folosite ca atribute.

Din perspectiva unui model-view-controller (MVC), fișierul FXML care conține descrierea interfeței de utilizator este view. Controller-ul este o clasă Java care este declarată ca controler pentru fișierul FXML, implementând opțional clasa Initializable,. Modelul constă din obiecte domeniu, definite pe partea de Java, pe care le conectați la view prin intermediul controller-ului.

În timp ce puteți utiliza FXML pentru a crea o interfață cu utilizatorul simpla, FXML este deosebit de util pentru interfețele utilizatorilor care au grafice mari cu scenă complexă, formulare, introducere de date sau animații complexe. FXML este, de asemenea, bine adaptat pentru definirea layout-urilor statice, cum ar fi formele, controale și tabelele. În plus, puteți utiliza FXML pentru a construi machete dinamice prin includerea de scripturi.

In plus pentru a oferi dezvoltatorilor web o abordare mai usoara în proiectarea interfețelor de utilizator, FXML oferă următoarele avantaje:

* Deoarece graficul de scenă este mai transparent în FXML, este ușor pentru o echipă de dezvoltare să creeze și să mențină o interfață de utilizator testabilă.
* FXML nu este un limbaj complicat; Nu este necesar să recompilați codul pentru a vedea modificările.
* Conținutul unui fișier FXML poate fi localizat pe măsură ce fișierul este citit. De exemplu, dacă un fișier FXML este încărcat utilizând locația en\_US, atunci acesta produce șirul "First Name" pentru un control bazat pe acest șir din resurse. Dacă locația este schimbată în fr\_FR și fișierul FXML este reîncărcat, atunci eticheta arată "Prénom".
* Același lucru nu este valabil și pentru codul Java, deoarece trebuie să actualizați manual conținutul fiecărui element al interfeței dvs. de utilizator, obținând o referință la acesta și chemând setterul corespunzător.
* Puteți utiliza FXML cu orice limbaj Java Virtual Machine (JVM), cum ar fi Java, Scala sau Clojure.

Așa cum unii dezvoltatori preferă să lucreze direct în codul XML, alți dezvoltatori preferă să utilizeze un instrument pentru a-și crea cod în XML. Același lucru este valabil și pentru FXML.

Dacă preferați să utilizați un instrument sau dacă doriți să creați un prototip rapid pentru a obține feedback, atunci ați putea prefera să utilizați JavaFX Scene Builder. Scenel Builder este un instrument de proiectare care generează codul sursă FXML în timp ce definiți interfața de utilizator pentru aplicația dvs. Scenel Builder vă poate ajuta să creați rapid un prototip pentru o aplicație interactivă care conectează componentele la logica aplicațiilor.

Deoarece Scenic Builder utilizează XML ca format de serializare, codul FXML produs este foarte clar și puteți edita în continuare fișierele FXML generate de Scene Builder în orice editor de text sau XML.

Eclipse IDE vă permite să deschideți fișiere FXML în JavaFX Scene Builder, cu condiția ca acesta să fie instalat pe computer. Această integrare Eclipse și Scene Builder oferă un avantaj suplimentar atunci când dezvoltați aplicații FXML.

JavaFX Scene Builder este un instrument vizual care permite utilizatorilor să proiecteze rapid interfețe de aplicații JavaFX fără a scrie cod. Utilizatorii pot trage și plasa componente UI într-o zonă de lucru, pot modifica proprietățile acestora, aplică stiluri, codul FXML este generat automat în fundal. Rezultatul este un fișier FXML care poate fi apoi combinat cu un proiect Java prin legarea UI la logica aplicației.

Proiectul Scenic Builder a fost creat folosind JavaFX de către Oracle și este open source în cadrul proiectului OpenJFX.

Oracle a furnizat fișiere binare, până la crearea Scene Builder v 2.0.Gluon a preluat distribuția de versiuni binare .

Spring Framework este un framework și un inversion of control container pentru platforma Java. Componentele principale ale framework-ului pot fi utilizate de orice aplicație Java, dar există extensii pentru construirea de aplicații web pe platforma Java EE. Deși cadrul nu impune nici un model specific de programare, acesta a devenit popular în comunitatea Java ca o alternativă , înlocuire sau adăugare la modelul Enterprise JavaBeans (EJB). Framework-ul este open source.

Partea centrala pentru Spring Framework este containerul său de inversare a controlului (IoC), care oferă un mijloc consistent de configurare și gestionare a obiectelor Java folosind reflecția. Conteinerul este responsabil pentru gestionarea ciclurilor de viață ale obiectelor specifice: crearea acestor obiecte, apelarea metodelor lor de inițializare și configurarea acestor obiecte prin wire.

Obiectele create de container sunt numite și obiecte bean. Containerul poate fi configurat prin încărcarea fișierelor XML sau detectarea anotatilor specifice Java în clasele de configurare. Aceste surse de date conțin definițiile de bean-uri care furnizează informațiile necesare creării bean-urilor.

Obiectele pot fi obținute fie prin căutare de dependență, fie prin injectare de dependență. Căutarea de dependență este un mod în care un apelant întreabă containerul pentru un obiect cu un anumit nume sau un anumit tip. Injecția de dependență este un mod în care containerul trece obiectele după nume în alte obiecte, fie prin constructori, fie prin proprietăți sau prin metode de tip factory.

În multe cazuri, nu este necesar să utilizați containerul atunci când utilizați alte părți ale framework-ului, deși utilizarea acestuia va face probabil aplicația mai ușor de configurat și personalizat. Framework-ul oferă un mecanism consecvent de configurare a aplicațiilor și se integrează cu aproape toate mediile Java, de la aplicații la scară mică până la aplicații de întreprinderi mari.

Containerul poate fi transformat într-un container parțial compatibil cu EJB 3. Unii critică Spring framework pentru nerespectarea standardelor. Cu toate acestea, SpringSource nu consideră conformitatea cu EJB ca fiind un obiectiv major și susține că Spring framework și containerul permit modele mai performante de programare . Acest lucru face ca acest cod să fie ușor de întreținut și să fie mai ușor de testat prin IoC.

Există multe avantaje ale Spring framework. Ele sunt după cum urmează:

* Șabloane predefinite. Spring framework oferă șabloane pentru tehnologiile JDBC, Hibernate, JPA etc. Deci nu este nevoie să scrieți prea mult cod. Acesta ascunde pașii de bază ai acestor tehnologii.
* Cuplajul liber. Aplicațiile care folosesc Spring sunt cuplate slab datorită injectării de dependențe.
* Ușor de testat. Injecția dependenței ușurează testarea aplicației. Aplicația EJB sau Struts cer serverului să ruleze aplicația, dar Spring framework nu necesită server.
* Dificultatea scazuta. Structura Spring este ușoară datorită implementării POJO-uri. Spring framework nu obligă programatorul să moștenească o anumita clasă sau să implementeze o interfață. De aceea se spune neinvazivă.
* Dezvoltare rapidă. Caracteristica Dependency Injection din Spring Framework și suportul pentru diverse cadre fac ca dezvoltarea aplicaților JavaEE să fie ușoară.
* Abstracție puternică. Oferă abstracție puternică în specificațiile JavaEE, cum ar fi JMS, JDBC, JPA și JTA.
* Sprijin declarative. Oferă suport declarativ pentru cache, validare, tranzacții și formatare.

Gradle este un sistem de automatizare a build-urilor, fiind open source. Se bazează pe conceptele Apache Ant și Apache Maven și introduce un limbaj specific domeniului Groovy (DSL) în locul formularului XML folosit de Apache Maven pentru declararea configurației proiectului. [2] Gradle utilizează un directed acyclic graph ("DAG") pentru a determina ordinea în care pot fi executate sarcini.

Gradle a fost proiectat pentru build-uri multi-proiect care poate deveni destul de mare și sprijină build-uri incrementale prin determinarea inteligentă a părților construite până la data actuală din arbore, astfel încât orice sarcină care depinde de aceste părți nu va trebui să fie reexecutata.

Pluginurile inițiale sunt concentrate în primul rând în jurul dezvoltării și implementării Java, Scala și Groovy.

Pe scurt, managementul dependențelor este alcătuit din două componente. În primul rând, Gradle trebuie să știe despre lucrurile pe care proiectul trebuie să le construiască sau să le execute, pentru a le găsi. Numim aceste fișiere primite dependințele proiectului. În al doilea rând, Gradle trebuie să construiască și să încarce lucrurile pe care le produce proiectul tău. Numim aceste fișiere de ieșire publicațiile proiectului