AUTOKER

Az alkalmazás a spring initializr oldal segítségével lett létrehozva Maven projektként Java nyelven és egy spring boot autókereskedéses webalkalmazásról beszélhetünk. Az oldal számos lehetőséget biztosít amely meg tudja könnyíteni a dolgunkat hiszen az alap adatok például osztály és csomag nevek megadása mellett lehetőség van a verzió és magának a nyelvnek a definiálására is. Ezeken felül úgynevezett függőségeket is hozzá tudunk adni a projektünkhöz. Ezek elengedhetetlenek ahhoz hogy a program a későbbiekben megfelelően működjön ugyanis egyes annotációkat, osztályokat vagy metódusokat nem ismeri fel a fordító program a hozzájuk tartozó .jar kiterjesztésű file-ok hiányában. Én személy szerint az alapértelmezett alkalmazás legenerálását is .jar kiterjesztésű fájlként szoktam legenerálni de arra is van lehetőségünk hogy ezt a .war kiterjesztéssel oldjuk meg.

A program célja: ------------------------------------

Egy autókereskedéses webalkalmazást valósítanak meg a kódsorok, amibe először be kell jelentkezni a megfelelő felhasználónévvel és jelszóval. Miután ez sikeresen megtörtént elénk tárul a weboldal nyitó felülete, amely tartalmaz néhány képet. Ezekre kattintva különböző funkcióknak tehetünk eleget, amelyek az autó hozzáadása, garázsok megtekintése illetve egy adott autó keresése.

Back-end

A JAR mint fájlformátum:

A Java archívum fájl egy csomagfájl formátum, amelyet általában sok Java osztályfájl és a kapcsolódó metaadatok, illetve erőforrások például szöveg vagy képek egyetlen fájlba aggregálására használunk terjesztés céljából. A JAR fájlok olyan archív fájlok, amelyek Java-specifikus jegyzékfájlt tartalmaznak. Ezek ZIP formátumra épülnek, és általában .jar fájlkiterjesztéssel rendelkeznek.

A JAR-fájl lehetővé teszi a Java futtatókörnyezetek számára, hogy egyetlen kérelemben hatékonyan telepítsenek egy teljes alkalmazást, beleértve annak osztályait és a hozzájuk tartozó erőforrásokat. A JAR fájlelemek tömöríthetők, lerövidítve a letöltési időt. A JAR-fájl tartalmazhat egy manifest fájlt is amit jegyzékfájlnak fordíthatunk. Ebben a bejegyzések magának a .jar kiterjesztésű fájlok használatát írják le.

Amikor a Java futtatókörnyezet betölti az JAR fájlokat, ellenőrizheti az hitelességet, és megtagadhatja az elvárt értékkel nem egyező osztályok betöltését. Ez megakadályozza, hogy rosszindulatú kódokat illesszünk be egy meglévő csomagba, és így hozzáférjenek a csomagra kiterjedő osztályokhoz és adatokhoz.

Egy futtatható Java program becsomagolható JAR-fájlba, a program által használt könyvtárakkal együtt. Egyes operációs rendszerek rákattintva közvetlenül futtathatják ezeket.

A WAR mint fájlformátum:

A szoftverfejlesztésben a WAR-fájl (Web Application Resource vagy Web Application ARchive olyan fájl, amelyet JAR-fájlok, JavaServer-oldalak, Java-szervletek, Java osztályok, XML-fájlok, címkekönyvtárak, statikus weboldalak és egyéb erőforrások, amelyek együtt egy webalkalmazást alkotnak.

A WAR-fájl ugyanúgy digitálisan aláírható, mint egy JAR-fájl, hogy mások meghatározhassák, honnan származik a forráskód. Egy WAR-fájlban speciális fájlok és könyvtárak találhatók. A WAR fájl /WEB-INF könyvtára egy web.xml nevű fájlt tartalmaz, amely meghatározza a webalkalmazás szerkezetét. Ha a webalkalmazás csak JSP-fájlokat szolgál ki, a web.xml fájl nem feltétlenül szükséges. Ha a webalkalmazás kiszolgáló kisalkalmazásokat használ, akkor a szervlet-tároló a web.xml segítségével állapítja meg, hogy az URL-kérés melyik szervlethez lesz irányítva.

A WAR fájlok előnyei:

* Webes alkalmazások egyszerű tesztelése és telepítése
* A telepített alkalmazás verziójának egyszerű azonosítása
* Minden Java EE tároló támogatja a WAR fájlokat
* Az MVC szerkezet támogatja a WAR fájlokat.

A WAR fájlok hátrányai:

Egyesek úgy vélik, hogy a WAR-fájlok használatával történő webes üzembe helyezés hátrányos, ha a dinamikus környezetekben kisebb változtatásokra van szükség a forráskódban. A forráskód minden módosítását újra kell csomagolni, és a fejlesztés során telepíteni kell. Ez nem igényli a webkiszolgáló leállítását, ha futásidejű telepítésre van beállítva.

Mielőtt nekiállunk egy adott programozási feladaton dolgozni azt fontos felismernünk, hogy melyik lenne számunkra az a leghasznosabb szoftver amely a projektünk menedzselésére szolgálhat. Az elérhető szoftverek közül kettőre fogok részletesebben kitérni amelyek az Apache Maven és a Gradle. Ezek részletes ismerete hozzásegíthet minket ahhoz hogy a számunkra megfelelőt válasszuk ki és a lehető leghatékonyabb munkát tudjuk végezni.

Apache Maven:

A Maven egy építési automatizálási eszköz, amelyet elsősorban Java projektekhez használnak de C#, Ruby, Scala és más nyelveken írt projektek készítésére és kezelésére is használható. A Maven projektnek az Apache Software Foundation ad otthont, ahol korábban a Jakarta Project része volt.

A Maven a szoftverkészítés két aspektusával foglalkozik: a szoftver felépítésével és függőségeivel. Csak a kivételeket kell megadni. Az XML-fájl leírja a készülő szoftverprojektet, annak más külső moduloktól és összetevőktől való függőségét, az összeállítási sorrendet, a könyvtárakat és a szükséges beépülő modulokat. Előre meghatározott célokat tartalmaz bizonyos jól meghatározott feladatok végrehajtásához, mint például a kód összeállítása és csomagolása. A Maven dinamikusan letölti a Java könyvtárakat és a Maven beépülő modulokat egy vagy több tárolóból, például a Maven 2 Central Repositoryból, és a helyi gyorsítótárban tárolja azokat. letöltött műtermékek helyi gyorsítótára frissíthető a helyi projektek által létrehozott műtermékekkel is. A nyilvános adattárak is frissíthetők.

A Maven plugin-alapú architektúrával készült, amely lehetővé teszi bármely szabványos bemeneten keresztül vezérelhető alkalmazás használatát. Az olyan alternatív technológiák, mint például a Gradle, mint összeállítási eszközök, nem támaszkodnak az XML-re, hanem megtartják a Maven által bevezetett kulcsfogalmakat. Az Apache Ivy-vel egy dedikált függőségi kezelőt is fejlesztettek, amely a Maven adattárakat is támogatja. Az Apache Maven támogatja a reprodukálható buildeket.

A Maven 3.0 átdolgozta az alapvető Project Builder infrastruktúrát, aminek eredményeként a POM fájl alapú reprezentációja leválasztott memória objektum reprezentáció. Ez kibővítette a Maven 3.0 bővítmények lehetőségét a nem XML alapú projektdefiníciós fájlok kihasználására. A Maven 3 első bétaverziójában bevezették a párhuzamos építési funkciót, amely konfigurálható számú magot hasznosít egy többmagos gépen, és különösen alkalmas nagy, többmodulos projektekhez.

A Maven projektek egy pom.xml fájlban tárolt Project Object Model (POM) segítségével vannak konfigurálva. Ez a POM csak a projekt egyedi azonosítóját (koordinátáit) és a JUnit keretrendszertől való függőségét határozza meg. Ez azonban már elegendő a projekt felépítéséhez és a projekthez kapcsolódó egységtesztek futtatásához. A Maven ezt úgy éri el, hogy a Maven alapértelmezett értékeket biztosít a projekt konfigurációjához.

* project home - A pom.xml fájlt és az összes alkönyvtárat tartalmazza.
* src/main/java - A projekt szállítható Java-forráskódját tartalmazza.
* src/main/resources - A projekt szállítható erőforrásait, például tulajdonságfájlokat tartalmazza.
* src/test/java - A projekt tesztelő Java forráskódját (például JUnit vagy TestNG teszteseteket) tartalmazza.
* src/test/resources - A teszteléshez szükséges erőforrásokat tartalmazza.

Az mvn parancscsomag lefordítja az összes Java fájlt, lefuttat minden tesztet, és a szállítható kódot és az erőforrásokat a target/my-app-1.0.jar fájlba csomagolja. A Maven használatával a felhasználó csak konfigurációt biztosít a projekthez, míg a konfigurálható beépülő modulok végzik el a projekt fordítását, a célkönyvtárak tisztítását, az egységtesztek futtatását, az API dokumentációk generálását és így tovább. Általában a felhasználóknak nem kell maguknak beépülő modulokat írniuk.

Project Object Model – POM:

A Projekt objektum modell egyetlen projekt összes konfigurációját biztosítja. Az általános konfiguráció lefedi a projekt nevét, tulajdonosát és más projektektől való függőségeit. Az építési folyamat egyes fázisai is konfigurálhatók, amelyek pluginként valósulnak meg. Például beállíthatjuk a fordító-bővítményt úgy, hogy a fordításhoz a Java 1.5-ös verzióját használja, vagy megadhatjuk a projekt csomagolását még akkor is, ha egyes egységtesztek meghiúsulnak. A nagyobb projekteket több modulra vagy alprojektre kell osztani, amelyek mindegyike saját POM-mal rendelkezik. Ezután írhatunk egy root POM-ot, amelyen keresztül egyetlen paranccsal lefordíthatjuk az összes modult. A POM-ok a konfigurációt más POM-októl is örökölhetik. Alapértelmezés szerint minden POM a Super POM-tól öröklődik. A Super POM alapértelmezett konfigurációt biztosít, például alapértelmezett forráskönyvtárakat, alapértelmezett beépülő modulokat és így tovább.

Plug-ins:

A Maven legtöbb funkciója beépülő modulokban található. Vannak Maven beépülő modulok az építéshez, teszteléshez, forrásvezérlés kezeléséhez, webszerver futtatásához, Eclipse projektfájlok generálásához és még sok máshoz. A beépülő modulok a pom.xml fájl plugins szakaszában kerülnek bevezetésre és konfigurálásra. Néhány alapvető bővítmény alapértelmezés szerint minden projektben megtalálható, és ésszerű alapértelmezett beállításokkal rendelkeznek.

A bővítmények a Maven kiterjesztésének elsődleges módja.

Életciklusok építése:

Az életciklus egy olyan elnevezés, amely segítségével parancsot adhatunk az adott cél végrehajtásához. A bővítmények által biztosított célok az életciklus különböző szakaszaihoz társíthatók.

A Maven szabványos fázisokkal is rendelkezik a projekt tisztításához és a projekt helyszínének létrehozásához. Ha a tisztítás az alapértelmezett életciklus része lenne, a projektet minden építéskor megtisztítanák. Ez egyértelműen nem kívánatos, ezért a tisztítás saját életciklust kapott. A szabványos életciklusok lehetővé teszik a projektben kezdő felhasználók számára, hogy pontosan építsenek, teszteljenek és telepítsenek minden Maven projektet az mvn install parancs kiadásával. Alapértelmezés szerint a Maven a POM-fájlt generált JAR- és WAR-fájlokba csomagolja.

Függőségek:

A Maven központi funkciója a függőségkezelés. A Maven függőséget kezelő mechanizmusa egy koordinátarendszer köré szerveződik, amely azonosítja az egyedi műtermékeket, például szoftverkönyvtárakat vagy modulokat. A fenti POM példa a JUnit koordinátákat a projekt közvetlen függőségeként hivatkozik. Egy projektnek, amelyhez például a Hibernate könyvtárra van szüksége, egyszerűen deklarálnia kell a Hibernate projekt koordinátáit a POM-ban. A Maven automatikusan letölti a függőséget és azokat a függőségeket, amelyekre magának a Hibernate-nek szüksége van (úgynevezett tranzitív függőségek), és eltárolja azokat a felhasználó helyi tárolójában. A Maven tervezése minden projektet bizonyos struktúrával és támogatott feladat-munkafolyamatokkal rendelkezőnek tekint. A Maven egy konvencióra támaszkodik a projektek meghatározását illetően, és azon munkafolyamatok listájára, amelyek általában minden projektben támogatottak.

Az egyetlen gépen kifejlesztett projektek a helyi adattáron keresztül függhetnek egymástól. helyi adattár egy egyszerű mappastruktúra, amely a letöltött függőségek gyorsítótáraként és a helyileg épített műtermékek központi tárolóhelyeként is működik. A Maven parancs mvn install létrehoz egy projektet, és elhelyezi a bináris fájljait a helyi tárolóban. Ezután más projektek használhatják ezt a projektet, ha megadják annak koordinátáit a POM-jukban.

A Java programozási nyelvet célzó, népszerű integrált fejlesztőkörnyezethez (IDE) léteznek kiegészítők, amelyek a Maven integrációját biztosítják az IDE felépítési mechanizmusával és forrásszerkesztő eszközeivel, lehetővé téve a Maven számára, hogy az IDE-n belülről fordítson projekteket, és beállítsa az osztályútvonalat is. Ezek a bővítmények lehetővé teszik a POM szerkesztését vagy a POM használatával a projekt teljes függőségi készletét közvetlenül az IDE-n belül. Az IDE-k egyes beépített funkciói elvesznek, ha az IDE már nem hajt végre fordítást. Sok IDE a Maven által preferált mappák hierarchiája helyett egy lapos projektkészlettel dolgozik.

Gradle:

A Gradle egy építési automatizálási eszköz többnyelvű szoftverfejlesztéshez. Felügyeli a fejlesztési folyamatot a fordítástól és a csomagolástól a tesztelésig, a telepítésig és a közzétételig. A támogatott nyelvek közé tartozik a Java, a C/C++ és a JavaScript. Ezenkívül statisztikai adatokat gyűjt a szoftverkönyvtárak használatáról világszerte. A Gradle az Apache Ant és az Apache Maven koncepcióira épít, és bevezet egy Groovy- és Kotlin-alapú tartományspecifikus nyelvet, szemben a Maven által használt XML-alapú projektkonfigurációval.

A Gradle irányított aciklikus gráfot használ a feladatok futtatásának sorrendjének meghatározására a függőségek kezelésével. Java virtuális gépen fut.

A Gradle-t többprojektes összeállításokhoz tervezték, amelyek nagyra nőhetnek. Egy sor összeállítási feladaton alapul, amelyek sorosan vagy párhuzamosan futhatnak. A növekményes összeállításokat a felépítési fa már naprakész részeinek meghatározása támogatja, így a csak ezektől a részektől függő feladatokat nem kell újra végrehajtani. Támogatja az összeállítási összetevők gyorsítótárazását is, potenciálisan megosztott hálózaton keresztül a Gradle Build Cache használatával. A szoftver bővíthető új funkciókkal és programozási nyelvekkel egy plugin alrendszerrel.

Miután kiválasztottuk a csomagfájl formátumot és a projektünket menedzselő szoftvert a legalapvetőbb kérdés következik. Ez nem más mint, hogy melyik az a programozási nyelv amelynek segítségével le akarjuk programozni az alkalmazásunkat. A hozzám legközelebb álló programozási nyelv a Java tehát ezzel dolgoztam.

A Java mint programozási nyelv:

A Java egy magas szintű, osztályalapú, objektum-orientált programozási nyelv, amelyet úgy terveztek, hogy a lehető legkevesebb megvalósítási függőséggel rendelkezzen. Ez egy általános célú programozási nyelv, amelynek célja, hogy a programozók egyszer írhassanak, bárhol lehessen futtatni ami azt jelenti, hogy a lefordított Java kód minden Java-t támogató platformon futhat újrafordítás nélkül. A Java-alkalmazásokat általában bájtkódra fordítják, amely bármely Java virtuális gépen (JVM) futhat, függetlenül az alapul szolgáló számítógép-architektúrától.

A Java szintaxisa hasonló a C-hez és a C++-hoz, de kevesebb alacsony szintű szolgáltatással rendelkezik, mint bármelyik. A Java futtatókörnyezet olyan dinamikus képességeket (például tükrözést és futásidejű kódmódosítást) biztosít, amelyek általában nem állnak rendelkezésre a hagyományos fordítási nyelveken. 2019-ben a Java volt az egyik legnépszerűbb programozási nyelv különösen a kliens-szerver webalkalmazások esetében.

A nyelvet eredetileg tölgynek hívták egy tölgyfa után. Később a projekt a Green nevet vette fel, és végül Java névre keresztelték, a Java kávéból, egy Indonéziából származó kávéfajtából. A Sun Microsystems 1996-ban adta ki az első nyilvános implementációt Java 1.0 néven. Meglehetősen biztonságos és konfigurálható biztonsággal, lehetővé tette a hálózat- és fájl-hozzáférési korlátozásokat. A nagy webböngészők beépítették a Java kisalkalmazások weblapokon belüli futtatásának lehetőségét, és a Java gyorsan népszerűvé vált. A J2EE technológiai és API-kat tartalmazott a vállalati alkalmazásokhoz, amelyek általában szerverkörnyezetben futnak, míg a J2ME mobilalkalmazásokra optimalizált API-kat tartalmazott. Az asztali verziót átnevezték J2SE-re.

A Java szoftverek a laptopoktól az adatközpontokon át a játékkonzolokon át a tudományos szuperszámítógépekig mindenen futnak.

A Java nyelv létrehozásának öt fő célja volt:

* Egyszerűnek, objektumorientáltnak és érthetőnek kell lennie.
* Kiszámíthatónak és biztonságosnak kell lennie.
* Hardver függetlennek és hordozhatónak kell lennie.
* Nagy teljesítménnyel kell végrehajtani a feladatokat.
* Többszálúnak és dinamikusnak kell lennie.

Kiadások:

Négy Java-kiadás definiált és támogatott, amelyek különböző alkalmazási környezeteket céloznak meg, és számos API-t szegmentált úgy, hogy azok valamelyik platformhoz tartozzanak. A platformok a következők:

* Java kártya intelligens kártyákhoz.
* Java Platform, Micro Edition (Java ME) – korlátozott erőforrásokkal rendelkező környezetek megcélzása.
* Java Platform, Standard Edition (Java SE) – munkaállomás-környezetek megcélzása.
* Java Platform, Enterprise Edition (Java EE) – nagy, elosztott vállalati vagy internetes környezeteket céloz meg.

A Java API-k osztályai külön csoportokba vannak szervezve, amelyeket csomagoknak nevezünk. Minden csomag tartalmaz egy sor kapcsolódó interfészt, osztályt, alcsomagot és kivételeket. Egy Personal Java nevű kiadást is biztosított volt, amelyet a későbbi, szabvány alapú Java ME konfigurációs-profil párosítások váltottak fel.

Végrehajtási rendszer:

Java JVM és bájtkód:

A Java egyik tervezési célja a hordozhatóság, ami azt jelenti, hogy a Java platformra írt programoknak hasonlóan kell futniuk a hardver és az operációs rendszer bármely kombinációján, megfelelő futásidejű támogatással. Ezt úgy érik el, hogy a Java nyelvi kódot egy Java bájtkódnak nevezett köztes reprezentációba fordítják, nem pedig közvetlenül architektúra-specifikus gépi kódra. A Java bájtkód utasításai hasonlóak a gépi kódhoz, de kifejezetten a gazdagép hardverére írt virtuális gép (VM) általi végrehajtásra készültek. A végfelhasználók általában a készülékükre telepített Java Runtime Environment-et (JRE) használják önálló Java-alkalmazásokhoz vagy webböngészőt Java-kisalkalmazásokhoz. A szabványos könyvtárak általános módot biztosítanak a gazdagép-specifikus funkciók, például a grafika, a szálkezelés és a hálózatok elérésére.

Az univerzális bájtkód használata egyszerűvé teszi a portolást. Azonban a bájtkód gépi utasításokká történő értelmezése miatt az értelmezett programok szinte mindig lassabban futnak, mint a natív végrehajtható fájlok. A bájtkódokat gépi kódokká futás közben fordító Just-in-time (JIT) fordítókat korai szakaszban vezették be. Maga a Java platformfüggetlen, és az adott platformhoz igazodik egy Java virtuális gép (JVM), amely lefordítja a Java bájtkódot a platform gépi nyelvére.

Teljesítmény:

A Java nyelven írt programok lassabbak és több memóriát igényelnek, mint a C++-ban írottak. A Java programok végrehajtási sebessége azonban jelentősen javult a Just-in-time verzió bevezetésével és a jobb kódelemzést támogató nyelvi szolgáltatásokkal például belső osztályok, StringBuilder osztály, opcionálisok stb.

Egyes platformok közvetlen hardveres támogatást kínálnak a Java számára. Vannak olyan mikrovezérlők, amelyek hardverben is képesek Java bájtkódot futtatni a szoftveres Java virtuális gép helyett és egyes ARM-alapú processzorok hardveres támogatást is kaphatnak a Java bájtkód végrehajtásához bár a támogatás többnyire megszűnt a jelenlegi megvalósításokban.

Automatikus memóriakezelés:

Java automatikus szemétgyűjtőt használ a memória kezelésére az objektum életciklusában. A programozó határozza meg, hogy mikor jönnek létre az objektumok, és a Java futási környezet felelős a memória helyreállításáért, ha az objektumok már nincsenek használatban. Ha nem marad hivatkozás egy objektumra, az elérhetetlen memória jogosulttá válik arra, hogy a szemétgyűjtő automatikusan felszabadítsa. A memóriaszivárgáshoz hasonló jelenség továbbra is előfordulhat, ha a programozó kódja hivatkozást tartalmaz egy olyan objektumra, amelyre már nincs szükség, általában akkor, ha a már nem szükséges objektumokat még használatban lévő tárolókban tárolják. Ha egy nem létező objektum metódusait hívjuk meg, null pointer kivételt dobódik.

A szemét begyűjtés bármikor megtörténhet. Ideális esetben akkor fordul elő, amikor egy program tétlen. Garantáltan aktiválódik, ha nincs elég szabad memória egy új objektum lefoglalásához és ez a program pillanatnyi leállását okozhatja. Az explicit memóriakezelés nem lehetséges a Java-ban. A memóriakezelési probléma megoldása nem mentesíti a programozót az egyéb erőforrások, például hálózati vagy adatbázis-kapcsolatok, fájlkezelők stb. megfelelő kezelésének terhe alól, különösen kivételek esetén.

Szintaxis:

A Java szinte kizárólag objektum-orientált nyelvként készült. Minden kód osztályokon belül van írva, és minden adatelem objektum, kivéve a primitív adattípusokat azaz egész számokat, lebegőpontos számokat, logikai értékeket és karaktereket, amelyek teljesítmény okokból nem objektumok. A többszörös öröklése nem támogatott az osztályok esetében, míg az interfészek esetében igen.

Három különböző stílusú megjegyzés létezik:

* egysoros stílus két perjellel ( // )
* egy többsoros stílus, amely /\*-val nyitódik és \*/-vel záródik
* valamint a Javadoc megjegyzésstílus, amely /\*\*-val nyitódik meg és \*/-vel záródik

Minden forrásfájlt a bennük lévő nyilvános osztályról kell elnevezni, hozzáfűzve a .java utótagot, például HelloWorldApp.java. Először bájtkódba kell fordítani egy Java fordító segítségével, egy .class utótagú fájl létrehozásával (ebben az esetben HelloWorldApp.class). Csak ezután lehet végrehajtani vagy elindítani. A Java forrásfájl csak egy nyilvános osztályt tartalmazhat, de több osztályt is tartalmazhat nem nyilvános hozzáférés-módosítóval és tetszőleges számú nyilvános belső osztályt. Ha a forrásfájl több osztályt tartalmaz, akkor egy osztályt (amelyet az osztály kulcsszó vezet be) nyilvánossá kell tenni (előtte a nyilvános kulcsszó), és el kell nevezni a forrásfájlt ezzel a nyilvános osztálynévvel.

A nyilvánosnak nem nyilvánított osztály bármely .java fájlban tárolható. A fordító létrehoz egy osztályfájlt a forrásfájlban meghatározott minden osztályhoz. Az osztályfájl neve az osztály neve, .class hozzáfűzve.

A public kulcsszó azt jelenti, hogy egy metódus meghívható más osztályok kódjából, vagy hogy egy osztályt az osztályhierarchián kívüli osztályok is használhatnak. Az osztályhierarchia annak a könyvtárnak a nevéhez kapcsolódik, amelyben a .java fájl található. Ezt hívják hozzáférési szint módosítónak. Az egyéb hozzáférési szint módosítók közé tartozik a private (egy olyan metódus, amely csak ugyanabban az osztályban érhető el) és a védett (amely lehetővé teszi az ugyanabból a csomagból származó kód elérését) kulcsszavakat. Ha egy kódrészlet privát metódusokhoz vagy védett metódusokhoz próbál hozzáférni, a JVM biztonsági kivételt dob.

A metódus előtt álló static kulcsszó egy statikus metódust jelöl, amely csak az osztályhoz kapcsolódik, és nem az adott osztály egyetlen példányához sem. Csak statikus metódusok hívhatók meg objektumra való hivatkozás nélkül. A statikus metódusok nem férhetnek hozzá olyan osztálytagokhoz, amelyek szintén nem statikusak. A nem statikus metódusok példánymetódusok, és működésükhöz egy osztály meghatározott példányára van szükség.

A void kulcsszó azt jelzi, hogy a fő metódus nem ad vissza semmilyen értéket a hívónak. Ha egy Java programnak hibakóddal kell kilépnie, akkor kifejezetten meg kell hívnia a System.exit() függvényt. A main metódusnév nem kulcsszó a Java nyelvben. Ez egyszerűen annak a módszernek a neve, amelyet a Java indító hív meg, hogy átadja a vezérlést a programnak. Egy Java-program több olyan osztályt is tartalmazhat, amelyeknek fő metódusai vannak, ami azt jelenti, hogy a virtuális gépnek kifejezetten meg kell adni, hogy melyik osztályból induljon el.

A fő metódusnak el kell fogadnia a String objektumok tömbjét. Megállapodás szerint args-ként hivatkoznak rá, bár bármely más törvényes azonosító név használható. A String[] args paraméter String objektumok tömbje, amely az osztálynak átadott argumentumokat tartalmazza. A fő paraméterek gyakran parancssoron keresztül kerülnek átadásra.

Speciális osztályok:

Applet

A Java kisalkalmazások olyan programok voltak, amelyeket más alkalmazásokba ágyaztak be, jellemzően a webböngészőben megjelenített weboldalakba.

Szervlet

A Java szervlet technológia egyszerű, konzisztens mechanizmust biztosít a webfejlesztők számára a webszerver funkcionalitásának bővítésére és a meglévő üzleti rendszerek elérésére. A szervletek szerveroldali Java EE összetevők, amelyek válaszokat generálnak az ügyfelektől érkező kérésekre. Ez legtöbbször HTML-oldalak generálását jelenti HTTP kérésekre válaszul, bár számos más szabványos szervlet osztály is elérhető, például a WebSocket kommunikációhoz.

A Java szervlet API-t bizonyos mértékig felváltotta (de továbbra is használatban van) két szabványos Java technológia webszolgáltatásokhoz:

* a Java API for RESTful Web Services (JAX-RS 2.0), amely hasznos az AJAX, JSON és REST szolgáltatásokhoz
* a Java API for XML Web Services (JAX-WS), amely hasznos a SOAP webszolgáltatásokhoz

Ezeknek az API-knak az alkalmazáskiszolgálókon vagy szervlettárolókon való tipikus megvalósítása szabványos szervletet használ a HTTP-kérésekkel és válaszokkal való összes interakció kezelésére, amely a tényleges üzleti logika webszolgáltatási metódusaira hagyatkozik.

JavaSzerver oldalak

A JavaServer Pages (JSP) kiszolgálóoldali Java EE összetevők, amelyek válaszokat, jellemzően HTML-oldalakat generálnak az ügyfelek HTTP-kéréseire. A JSP-k Java kódot ágyaznak be egy HTML-oldalba a speciális <% és %> elválasztók használatával. A JSP-t egy Java szervlet-be, egy önálló Java-alkalmazásba fordítják, az első hozzáféréskor. Ezt követően a generált szervlet létrehozza a választ.

Swing alkalmazás

A Swing egy grafikus felhasználói felület könyvtár a Java SE platformhoz. A Swing csatlakoztatható megjelenés és érzetrendszere révén más megjelenést és érzetet is megadhat.

JavaFX alkalmazás

A JavaFX egy szoftverplatform asztali alkalmazások, valamint gazdag webalkalmazások létrehozására és szállítására, amelyek számos eszközön futhatnak. A JavaFX a Java SE szabványos grafikus felhasználói felületének könyvtáraként a Swinget helyettesíti. A JavaFX támogatja az asztali számítógépeket és a webböngészőket Microsoft Windows, Linux és macOS rendszeren viszont nem támogatja a natív operációs rendszer megjelenését és működését.

Generikus

A generikusok bevezetése előtt minden változódeklarációnak meghatározott típusúnak kellett lennie. Ez például a konténerosztályok esetében probléma, mert nincs egyszerű módja egy olyan tároló létrehozásának, amely csak meghatározott típusú objektumokat fogad el. A konténer vagy egy osztály vagy interfész összes altípusán működik, általában az Object-en, vagy minden egyes tárolt osztályhoz más tároló osztályt kell létrehozni. Az generikusok lehetővé teszik a fordítási idejű típusellenőrzést anélkül, hogy sok konténerosztályt kellene létrehozni, amelyek mindegyike majdnem azonos kódot tartalmaz. A hatékonyabb kód engedélyezése mellett bizonyos futásidejű kivételek előfordulását a fordítás idejű hibák kibocsátásával akadályozzák meg. Ha a Java megakadályozza az összes futásidejű típusú hiba (ClassCastExceptions) előfordulását, akkor típusbiztos lenne.

Osztálykönyvtárak:

A Java Class Library egy szabványos könyvtár, amelyet az alkalmazások fejlesztésének támogatására fejlesztettek ki Java nyelven.

Egy program fejlesztése előtt amit még fontos megfontolnunk az az hogy melyik az a számunkra kényelmes és kézre álló fejlesztői környezet aminek a segítségével a leghatékonyabb munkát tudjuk biztosítani. Én amióta programozással foglalkozom az Eclipset használom mert ez volt számomra a legszimpatikusabb. Próbáltam más környezetekben is fejleszteni például a Visual Studio Code-ban de ez nem nyerte el a tetszésemet.

Eclipse mint szoftver:

Az Eclipse a számítógépes programozásban használt integrált fejlesztői környezet (IDE). Tartalmaz egy alap munkaterületet és egy bővíthető beépülő rendszert a környezet testreszabásához. Ez a második legnépszerűbb IDE a Java fejlesztéshez. Az Eclipse többnyire Java nyelven íródott, és elsődleges felhasználása Java alkalmazások fejlesztésére szolgál, de más programozási nyelveken is használható alkalmazások fejlesztésére beépülő modulokon keresztül, például C, C++, Python, stb. Használható dokumentumok fejlesztésére is. A fejlesztői környezetek közé tartozik többek között az Eclipse Java fejlesztői eszközök (JDT) Java és Scala számára, az Eclipse CDT a C/C++ számára és az Eclipse PDT a PHP számára.

A Java fejlesztői eszközöket tartalmazó Eclipse szoftverfejlesztő készlet (SDK) a Java fejlesztők számára készült. A felhasználók bővíthetik lehetőségeiket az Eclipse Platformhoz írt beépülő modulok telepítésével, például fejlesztői eszközkészletekkel más programozási nyelvekhez, és saját beépülő moduljaikat írhatják, és járulhatnak hozzá.

Az Eclipse szoftverfejlesztő készlet (SDK) egy ingyenes és nyílt forráskódú szoftver, amelyet az Eclipse Public License feltételei szerint adnak ki.

Engedélyezés:

Az Eclipse Public License (EPL) az az alapvető licenc, amely alapján az Eclipse projekteket kiadják. Az Eclipse eredetileg a Common Public License alatt jelent meg, de később újra licencelték az Eclipse Public License alatt. A Free Software Foundation azt mondta, hogy mindkét licenc szabad szoftverlicenc.

Elnevezése:

A név a Microsoft Visual Studio volt, amit az Eclipse-nek kellett elhomályosítania. Az Eclipse különböző verziói különböző tudományos vonatkozású neveket kaptak. A Callisto, Europa és Ganymedes nevét viselő változatokat, amelyek a Jupiter holdjai, követte a Galileiról, a holdak felfedezőjéről elnevezett változat. Ezeket követte két nap-témájú név, a görög mitológiából származó Héliosz és az Indigo, a szivárvány hét színének egyike.

Szerkezete:

Az Eclipse beépülő modulokat használ az összes funkció biztosításához a futásidejű rendszeren belül. Amellett, hogy lehetővé teszi az Eclipse Platform kiterjesztését más programozási nyelvekkel, például C és Python használatával, a beépülő keretrendszer lehetővé teszi az Eclipse Platform számára, hogy együttműködjön hálózati alkalmazásokkal, mint például a telnet és adatbázis-kezelő rendszerekkel. A beépülő architektúra támogatja a kívánt kiterjesztések írását a környezetbe, például a konfigurációkezeléshez. Egy kis futásidejű kernel kivételével az Eclipse-ben minden beépülő modul. Így minden kifejlesztett beépülő modul ugyanúgy integrálható az Eclipse-be, mint a többi beépülő modul.

Példák a beépülő modulokra: az Unified Modeling Language (UML), a Sequence és más UML diagramok, a DB Explorer beépülő modulja és még sok más.

Alkalmazás életciklus-kezelés:

Az alkalmazások életciklus-kezeléséhez (ALM-Application Lifecycle Management) és az Eclipse feladatkezeléséhez szükség van egy Mylyn nevű opcionális összetevőre, amely a feladatközpontú felület nyílt forráskódú megvalósítása. API-t biztosít a feladatközpontú felületet beágyazó eszközökhöz. A szoftverfejlesztők számára ez segíti a fejlesztőket abban, hogy hatékonyan dolgozhassanak számos különböző feladattal például hibákkal, hibajelentésekkel vagy új szolgáltatásokkal. A feladatok be vannak építve a Mylynbe. Az összes integrált feladatnál a Mylyn figyeli a felhasználói tevékenységet, és megpróbálja azonosítani az adott feladathoz kapcsolódó információkat. Ezt a feladatkörnyezetet használja arra, hogy az Eclipse felhasználói felületet a kapcsolódó információkra összpontosítsa.

Spring keretrendszer:

A Spring Framework egy alkalmazás-keretrendszer és a vezérlőkonténer inverziója a Java platformhoz. A keretrendszer alapvető funkcióit bármely Java-alkalmazás használhatja, de vannak kiterjesztések webalkalmazások készítéséhez a Java EE (Enterprise Edition) platformon. A Spring Framework nyílt forráskódú.

A Spring 4.0 jelentős fejlesztései közé tartozik a Java SE (Standard Edition) 8, a Groovy 2, a Java EE 7 és a WebSocket egyes aspektusainak támogatása.

Modulok:

* Spring Core Container: ez a Spring alapmodulja, és rugalas konténereket biztosít (BeanFactory és ApplicationContext)
* Aspektus-orientált programozás: lehetővé teszi az átfogó szempontok megvalósítását
* Hitelesítés és engedélyezés: konfigurálható biztonsági folyamatok, amelyek egy sor szabványt, protokollt, eszközt és gyakorlatot támogatnak a Spring Security alprojekten keresztül
* Konfigurációs konvenció: a Spring Roo modulban egy gyors alkalmazásfejlesztési megoldást kínálnak a Spring-alapú vállalati alkalmazásokhoz
* Adatelérés: munka relációs adatbázis-kezelő rendszerekkel Java platformon Java Database Connectivity (JDBC) és objektumrelációs leképezési eszközökkel, valamint NoSQL adatbázisokkal
* A vezérlőkonténer megfordítása: az alkalmazáskomponensek konfigurálása és a Java objektumok életciklus-kezelése, főként függőségi befecskendezéssel
* Üzenetkezelés: az üzenetfigyelő objektumok deklaratív regisztrációja az üzenetsorokból a Java Message Service (JMS) szolgáltatáson keresztüli transzparens üzenetfogyasztás érdekében, az üzenetküldés javítása a szabványos JMS API-kon keresztül
* Model-view-controller: HTTP- és szervlet-alapú keretrendszer, amely a webalkalmazások és a RESTful (reprezentációs állapotátvitel) webszolgáltatások kibővítéséhez és testreszabásához nyújt kapcsokat
* Távoli hozzáférési keretrendszer: Java objektumok deklaratív távoli eljáráshívás (RPC) stílusú rendezése Java távoli metódushívás (RMI), CORBA-t (Common Object Request Broker Architecture) és HTTP-alapú protokollokat, beleértve a webszolgáltatásokat SOAP (Simple Object Access) támogató hálózatokon keresztül
* Tranzakciókezelés: egyesíti a tranzakciókezelő API-kat, és koordinálja a Java objektumok tranzakcióit
* Távkezelés: Java objektumok deklaratív megjelenítése és kezelése helyi vagy távoli konfigurációhoz Java Management Extensions (JMX) segítségével
* Tesztelés: támogató osztályok egységtesztek és integrációs tesztek írásához

A vezérlőtartály megfordítása (függőségi befecskendezés):

A Spring Framework központi eleme az inverziós vezérlő (IoC) konténer, amely konzisztens eszközt biztosít a Java objektumok konfigurálásához és kezeléséhez a tükrözés segítségével. A tároló felelős bizonyos objektumok objektum-életciklusainak kezeléséért:

* létrehozza ezeket az objektumokat
* meghívja inicializálási metódusaikat
* konfigurálja ezeket az objektumokat összekapcsolással

A tároló által létrehozott objektumokat felügyelt objektumoknak vagy komponenseknek is nevezik. A konténer konfigurálható XML (Extensible Markup Language) fájlok betöltésével vagy adott Java megjegyzések észlelésével a konfigurációs osztályokon. Ezek az adatforrások tartalmazzák azokat a bean definíciókat, amelyek a bean-ek létrehozásához szükséges információkat biztosítják.

Az objektumok függőségi kereséssel vagy függőségi injektálással is megszerezhetők. A függőségi keresés egy olyan minta, amelyben a hívó egy adott nevű vagy adott típusú objektumot kér a tárolóobjektumtól. A függőségi befecskendezés egy olyan minta, amelyben a tároló név szerint adja át az objektumokat más objektumoknak, konstruktorok, tulajdonságok vagy gyári metódusok segítségével.

Sok esetben nem kell használni a tárolót a Spring Framework más részeinek használatakor, bár ennek használata valószínűleg megkönnyíti az alkalmazás konfigurálását és testreszabását. A Spring tároló konzisztens mechanizmust biztosít az alkalmazások konfigurálásához, és szinte az összes Java-környezetbe integrálható, a kisméretű alkalmazásoktól a nagyvállalati alkalmazásokig.

Aspektus-orientált programozási keretrendszer:

A Spring keretrendszer saját szempont-orientált programozási (AOP) keretrendszerrel rendelkezik, amely kezeli a szempontokat. A különálló AOP-keretrendszer létrehozásának motivációja abból a meggyőződésből fakad, hogy lehetővé kell tenni az alapvető AOP-szolgáltatásokat anélkül, hogy túlzottan bonyolult lenne a tervezésben, a megvalósításban vagy a konfigurációban. A Spring AOP keretrendszer teljes mértékben kihasználja a Spring tároló előnyeit.

A Spring AOP-t úgy alakították ki, hogy képes legyen együttműködni a Spring keretrendszeren belüli horizontális problémákkal. A tároló által létrehozott és konfigurált bármely objektum gazdagítható a Spring AOP segítségével.

A Spring Framework a Spring AOP-t használja belsőleg a tranzakciókezeléshez, a biztonsághoz, a távoli hozzáféréshez és a JMX-hez.

Adathozzáférési keretrendszer:

A Spring adatelérési keretrendszere megoldja azokat a gyakori nehézségeket, amelyekkel a fejlesztők szembesülnek, amikor adatbázisokkal dolgoznak az alkalmazásokban. Támogatás biztosított az összes népszerű adatelérési keretrendszerhez Java nyelven: JDBC, Hibernate, Jakarta Persistence API (JPA), stb. Az összes támogatott keretrendszerhez a Spring biztosítja ezeket a funkciókat

* Erőforrás-kezelés – adatbázis-erőforrások automatikus megszerzése és felszabadítása
* Kivételkezelés – az adathozzáféréssel kapcsolatos kivételek lefordítása egy Spring adathozzáférési hierarchiába
* Tranzakciós részvétel – átlátható részvétel a folyamatban lévő tranzakciókban
* Erőforrás kibontása – adatbázis-objektumok lekérése a kapcsolatkészlet-burkolókból
* Absztrakció bináris nagy objektumok (BLOB) és karakteres nagy objektumok (CLOB) kezelésére

A Spring tranzakciókezelésével együtt az adathozzáférési keretrendszer rugalmas absztrakciót kínál az adathozzáférési keretrendszerekkel való munkavégzéshez. A Spring Framework az egyetlen olyan Java keretrendszer, amely felügyelt adatelérési környezetet kínál alkalmazáskiszolgálón vagy tárolón kívül.

Tranzakciókezelés:

A Spring tranzakciókezelési keretrendszere absztrakciós mechanizmust hoz a Java platformra. Absztrakciója képes:

* helyi és globális tranzakciókkal végzett munka (a helyi tranzakcióhoz nincs szükség alkalmazáskiszolgálóra)
* beágyazott tranzakciókkal dolgozik
* mentési pontokkal dolgozik
* a Java platform szinte minden környezetében működik

A Spring adatelérési keretrendszerével együtt – amely integrálja a tranzakciókezelési keretrendszert – lehetővé válik a tranzakciós rendszer felállítása konfiguráción keresztül anélkül, hogy JTA-ra vagy EJB-re kellene hagyatkozni. A tranzakciós keretrendszer az üzenetküldő és gyorsítótárazó motorokkal is integrálható.

Modell–nézet–vezérlő keretrendszer:

A Spring Framework saját modell-nézet-vezérlő (MVC) webalkalmazás-keretrendszerrel rendelkezik, amelyet eredetileg nem terveztek. Amikor viszont észrevették, hogy nem volt kellően elválasztva a prezentáció és a kéréskezelési réteg, valamint a kéréskezelési réteg és a modell akkor ennek megoldására implementálták a modell–nézet–vezérlő keretrendszert.

A Spring MVC által meghatározott legfontosabb interfészeket és azok felelőssége:

* Vezérlő: a Model és a View között jön a bejövő kérések kezelésére és a megfelelő válaszra való átirányításra. A vezérlő leképezi a http kérést a megfelelő metódusokhoz. Kapuként működik, amely irányítja a bejövő információkat. Vált a modell vagy a nézet között.
* HandlerAdapter: a bejövő kéréseket kezelő objektumok végrehajtása
* HandlerInterceptor: a bejövő kérések elfogása összehasonlítható, de nem egyenlő a Servlet-szűrőkkel
* HandlerMapping: a bejövő kéréseket kezelő objektumok (kezelők) kiválasztása a kéréseken belüli vagy külső tulajdonságok vagy feltételek alapján
* LocaleResolver: egy egyéni felhasználó területi beállításának feloldása és opcionális mentése
* MultipartResolver: megkönnyíti a fájlfeltöltésekkel való munkát a bejövő kérések becsomagolásával
* Nézet: felelős a válasz visszaküldéséért az ügyfélnek
* ViewResolver: Nézet kiválasztása a nézet logikai neve alapján

Távoli hozzáférési keretrendszer:

A Spring Remote Access keretrendszere egy absztrakció a Java platformon elérhető különféle RPC (távoli eljáráshívás) alapú technológiákkal való munkavégzéshez, mind az ügyfélkapcsolathoz, mind a szervereken lévő objektumok rendezéséhez. A keretrendszer által kínált legfontosabb funkció az, hogy a lehető legnagyobb mértékben megkönnyítse e technológiák konfigurálását és használatát a vezérlés és az AOP inverziójának kombinálásával.

Spring Boot - Gyors alkalmazásfejlesztés:

A Spring Boot a Spring hagyományos, konfiguráláson felüli megoldása önálló, éles szintű Spring alapú alkalmazások létrehozására. A legtöbb Spring Boot alkalmazásnak nagyon kevés Spring konfigurációra van szüksége.

Főbb jellemzői:

* Készítsen önálló Spring alkalmazásokat
* A Tomcat vagy a Jetty közvetlenül beágyazása (nem kell WAR fájlokat telepíteni)
* A Maven/Gradle konfiguráció leegyszerűsítéséhez biztosítson projektobjektum-modelleket (POM)
* A Spring automatikus konfigurálása, amikor csak lehetséges
* Biztosítson gyártásra kész funkciókat, például mérőszámokat, állapotellenőrzéseket és külső konfigurációkat
* Egyáltalán nincs kódgenerálás, és nincs szükség XML-konfigurációra
* Sima integráció és támogatja az összes vállalati integrációs mintát

Az alkalmazásban fellelhetőek különböző képek is amelyek a program egyszerűbb átláthatóságát hivatottak szolgálni. A felhasznált képek kiterjesztése .jpg formátumú. Ezt a kiterjesztést a JPEG típus kód foglalja magába.

JPEG:

A JPEG a digitális képek veszteséges tömörítésének általánosan használt módszere, különösen a digitális fényképezéssel előállított képek esetében. A tömörítés mértéke állítható, így választható kompromisszum a tárolás mérete és a képminőség között. A JPEG általában 10:1-es tömörítést ér el, a képminőségben alig érzékelhető veszteséggel. 1992-es bevezetése óta a JPEG a legszélesebb körben használt képtömörítési szabvány a világon, és a legszélesebb körben használt digitális képformátum, 2015-től kezdve naponta több milliárd JPEG-képet készítettek.

A "JPEG" kifejezés a Joint Photography Experts Group rövidítése, amely 1992-ben hozta létre a szabványt. A JPEG nagymértékben felelős a digitális képek és digitális fényképek elterjedéséért az interneten, majd később a közösségi médiában.

A JPEG-tömörítést számos képfájl-formátumban használják. A JPEG/Exif a digitális fényképezőgépek és más fényképészeti képrögzítő eszközök által használt leggyakoribb képformátum. A JPEG/JFIF mellett ez a legelterjedtebb formátum a fényképes képek tárolására és továbbítására a világhálón. Ezeket a formátumváltozatokat gyakran nem különböztetik meg egymástól, és egyszerűen JPEG-nek nevezik őket. A JPEG fájlok általában .jpg vagy .jpeg kiterjesztéssel rendelkeznek.

Tipikus használat:

A JPEG tömörítési algoritmus a legjobban olyan fényképeken működik, amelyek valósághű jeleneteket tartalmaznak, egyenletes tónus és színváltozatokkal. Webhasználat esetén, ahol a képhez felhasznált adatmennyiség csökkentése fontos a reszponzív megjelenítéshez, a JPEG tömörítési előnyei népszerűvé teszik a JPEG-et. A JPEG/Exif a digitális fényképezőgépek által leggyakrabban mentett formátum is.

A JPEG azonban nem alkalmas vonalas rajzokhoz és más szöveges vagy ikonikus grafikákhoz, ahol a szomszédos képpontok közötti éles kontraszt észrevehető eltéréseket okozhat. Az ilyen képeket jobb veszteségmentes grafikai formátumban, például TIFF, GIF, PNG vagy nyers képformátumban menteni. A JPEG szabvány veszteségmentes kódolási módot is tartalmaz, de ezt a módot a legtöbb termék nem támogatja.

Mivel a JPEG tipikus használata veszteséges tömörítési módszer, amely csökkenti a képhűséget, ezért nem alkalmas képalkotási adatok pontos reprodukálására például egyes tudományos és orvosi képalkotó alkalmazások és bizonyos technikai képfeldolgozási munkákra.

A JPEG szintén nem alkalmas a többszörösen szerkesztett fájlokhoz, mivel a képminőség minden egyes újratömörítésekor elveszik, különösen ha a képet levágják vagy eltolják, vagy ha a kódolási paramétereket megváltoztatják. A szekvenciális és ismétlődő szerkesztés során a képinformációk elvesztésének megelőzése érdekében az első szerkesztés veszteségmentes formátumban menthető, majd ebben a formátumban szerkeszthető, majd végül JPEG formátumban közzétehető terjesztés céljából.

Front-end

HTML:

A HyperText Markup Language vagy HTML a szabványos jelölőnyelv a webböngészőben való megjelenítésre tervezett dokumentumokhoz. Ezt olyan technológiák segíthetik, mint a Cascading Style Sheets (CSS) és a szkriptnyelvek, például a JavaScript.

A webböngészők HTML dokumentumokat fogadnak egy webszerverről vagy helyi tárolóról, és a dokumentumokat multimédiás weboldalakká alakítják. A HTML szemantikailag írja le a weboldal szerkezetét, és eredetileg a dokumentum megjelenésére utaló jeleket tartalmazott.

A HTML elemek a HTML oldalak építőkövei. A HTML-konstrukciók segítségével képek és egyéb objektumok, például interaktív űrlapok beágyazhatók a megjelenített oldalba. A HTML lehetőséget biztosít strukturált dokumentumok létrehozására a szöveg strukturális szemantikájának, például címsorok, bekezdések, listák, hivatkozások, idézetek és egyéb elemek jelölésével. A HTML elemeket címkék határolják, szögletes zárójelekkel írva. Az olyan címkék, mint az <img /> és az <input />, közvetlenül vezetnek be tartalmat az oldalra. Más címkék, például a <p>, körülvesznek és információt nyújtanak a dokumentum szövegéről, és más címkéket is tartalmazhatnak alelemként. A böngészők nem jelenítik meg a HTML címkéket, hanem az oldal tartalmának értelmezésére használják őket.

A HTML képes beágyazni olyan programokat, amelyek szkriptnyelven, például JavaScripten íródnak, ami befolyásolja a weboldalak viselkedését és tartalmát. A CSS beépítése meghatározza a tartalom megjelenését és elrendezését.

Jelölés:

A HTML-jelölés több kulcsfontosságú összetevőből áll, beleértve a címkéket és attribútumaikat, a karakteralapú adattípusokat, a karakterhivatkozásokat és az entitáshivatkozásokat. A HTML-címkék leggyakrabban olyan párokban állnak rendelkezésre, mint a <h1> és a </h1>, bár némelyik üres elemeket képvisel, ezért nincsenek párosítva, például az <img>. Egy ilyen párban az első címke a kezdőcímke, a második a záró címke.

A <html> és </html> közötti szöveg írja le a weboldalt, a <body> és </body> közötti szöveg pedig a látható oldal tartalma. A <title>Ez egy cím</title> jelölőszöveg határozza meg a böngésző lapjain és ablakcímein megjelenő böngészőoldal címét, a <div> címke pedig az oldal felosztását határozza meg az egyszerű stílus érdekében. A <head> és </head> között egy <meta> elem használható a weboldal metaadatainak meghatározására.

A nyelvben fellelhető elemek:

* Címsorok
* Sortörések
* Bemenetek
* Hozzászólások
* Attribútumok

CSS:

A Cascading Style Sheets (CSS) egy stíluslapnyelv, amelyet egy jelölőnyelven, például HTML-ben vagy XML-ben írt dokumentumok megjelenítésének leírására használnak. A CSS a World Wide Web egyik sarokköve a HTML és a JavaScript mellett.

A CSS-t úgy tervezték, hogy lehetővé tegye a tartalom és a megjelenítés elkülönítését, beleértve az elrendezést, a színeket és a betűtípusokat. Ez a szétválasztás javíthatja a tartalom hozzáférhetőségét:

* nagyobb rugalmasságot és ellenőrzést biztosítanak a megjelenítési jellemzők specifikációjában
* lehetővé teszi több weboldal formázásának megosztását a megfelelő CSS-nek egy külön .css fájlban történő megadásával, ami csökkenti a szerkezeti tartalom bonyolultságát és ismétlődését
* engedélyezi a .css fájl gyorsítótárazását a fájlt megosztó oldalak közötti oldalbetöltési sebesség és a formázás javítása érdekében

A formázás és a tartalom szétválasztása azt is lehetővé teszi, hogy ugyanazt a jelölőoldalt különböző stílusban mutassák be a különböző megjelenítési módszerekhez, például képernyőn, nyomtatásban, hanggal. A CSS-nek vannak szabályai az alternatív formázásra is, ha a tartalomhoz mobileszközről férünk hozzá.

Szintaxisa:

A CSS egyszerű szintaxissal rendelkezik, és számos angol kulcsszót használ a különböző stílustulajdonságok nevének megadásához.

A CSS-ben a választók deklarálják, hogy a jelölés mely részére vonatkozik egy stílus a címkék és attribútumok egyeztetésével magában a jelölésben. A kiválasztók a következőkre vonatkozhatnak:

* meghatározott típusú összes elemre, például a második szintű fejlécek h2
* attribútum által meghatározott elemekre, különösen:
* id: a dokumentumon belül egyedi azonosító, amelyet a választó nyelvében hash előtag jelöl
* osztály: egy azonosító, amely egy dokumentumban több elemet is képes megjegyzésekkel ellátni, pont előtaggal jelölve
* elemekre attól függően, hogy hogyan helyezkednek el a dokumentumfában a többihez képest

Az osztályok és azonosítók megkülönböztetik a kis- és nagybetűket, betűkkel kezdődnek, és alfanumerikus karaktereket, kötőjeleket és aláhúzásjeleket tartalmazhatnak. Egy osztály bármely elem tetszőleges számú példányára vonatkozhat. Az azonosító csak egyetlen elemre alkalmazható.

A választók sokféleképpen kombinálhatók a nagy specifikusság és rugalmasság elérése érdekében. Több választó egyesíthető egy szóközzel elosztott listában, hogy hely, elemtípus, azonosító, osztály vagy ezek kombinációja szerint határozza meg az elemeket.

Ezen kívül a nyelvben fellelhetőek további különböző speciális elemek például nyilatkozat blokkok, hosszúság egységek stb.