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 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.