1. Szoftverfejlesztés

**Mi a szoftverfejlesztés?**

* Definíció (SEVOCAB):
  + Tudományos és technológiai tudás, módszerek és tapasztalat szisztematikus alkalmazása szoftverek tervezéséhez, megvalósításához, teszteléséhez és dokumentálásához.
* Definíció (Sommerville):
  + A szoftverfejlesztés (software engineering) mérnöki tudományág, mely a szoftverek létrehozásának valamennyi kérdésével foglalkozik a rendszer specifikáció korai szakaszaitól kezdve a rendszer használatba vétel után karbantartásáig.

**Fogalmak**

***Programkönyvtár***

Definíció (SWEBOOK): Egy programkönyvtár szoftverek és kapcsolódó dokumentáció olyan együttese, melyet abból a célból terveztek, hogy a szoftverfejlesztést, használatot és karbantartást segítse.

Egy könyvtár erőforrásokat tartalmaz, mint adatok és kód, és egy jól meghatározott interfésze (API-ja) van.

***Alkalmazásprogramozási interfész (API)***

Definíció (SWEBOOK): Egy alkalmazásprogramozási interfész (API) egy könyvtár vagy egy keretrendszer által a felhasználók számára alkalmazások írásához exportált és elérhető szignatúrákat jelenti. A szignatúrák mellett egy API mindig közléssel kell, hogy szolgáljon a programok hatásáról és/vagy viselkedéséről.

Definíció (SEVOCAB): Szoftverkomponens, mely lehetővé teszi, hogy szoftveralkalmazások egymással kommunikáljanak.

***Keretrendszer***

Definíció (SWEBOOK): Egy keretrendszer egy részlegesen befejezett szoftverrendszer, mely bizonyos kiterjesztések (például plugin-ek) megfelelően történő példányosításával terjeszthető ki.

Definíció (Schmidt et al.): Egy keretrendszer szoftvertermékek (például osztályok, objektumok és komponensek) olyan integrált együttese, melyek abból a célból működnek együtt, hogy egy újrafelhasználható architektúrát biztosítsanak hasonló alkalmazások egy családjához.

Egy keretrendszer egy **félkész alkalmazás**. A fejlesztők a keretrendszer **újrafelhasználható komponenseinek kiterjesztésével és testreszabásával** alkotnak teljes alkalmazásokat.

A keretrendszerek egy **csontváz architektúrát biztosítanak** az alkalmazásokhoz. A keretrendszerek jellemzően mintákon alapulnak, mint az architekturális minták és tervezési minták.

Példák:

* C/C++: Boost.Test, GoogleTest, Qt
* Java: Hibernate ORM, JUnit 5, Mockito, Play Framework, Spring Framework
* JavaScript: Angular, Express, Jest, Mocha, Next.js, Vue.js
* Python: Django, Flask, unittest

1. Java technológiai áttekintés

**Fogalmak**

***Java (programozási nyelv)***

Általános célú, konkurens, osztály alapú, objektumorientált programozási nyelv.

***Java virtuális gép (Java virtual machine) (JVM)***

Egy saját utasításkészlettel rendelkező és vezérelhető absztrakt számítógép, mely semmit sem tud a Java programozási nyelvről, hanem csak egy speciális bináris formátumot ismer, a class állományformátumot. A Java virtuális gép a Java platform sarokköve.

***Java Platform***

A Java platform egy olyan szoftver platform, mely más hardver platformokon fut, és melynek az alábbi két fő komponense van: Java virtuális gép (JVM) és Java alkalmazásprogramozási interfész (API).

***Java Platform, Standard Edition (Java SE)***

Java platform asztali és szerver számítógépekre történő alkalmazásfejlesztéshez.

***Java Platform, Enterprise Edition (Java EE)***

API-t és futtató környezetet biztosít többrétegű, skálázható, megbízható és biztonságos szerver oldali alkalmazások fejlesztéséhez és futtatásához. A Java SE platformra épül. Később Jakarta EE lett belőle.

***Java Runtime Environment (JRE)***

A Java programozási nyelven írt programok végrehajtásához szükséges állományokat tartalmazza: egy virtuális gépet, osztálykönyvtárakat és további kapcsolódó állományokat.

***Java Development Kit (JDK)***

Fejlesztői környezet alkalmazások és komponensek a Java programozási nyelven történő készítéséhez.

***OpenJDK***

„Az OpenJDK közösség fejlesztők egy olyan társulása, akik a JCP által definiált Java SE jelenlegi és jövőbeli verzióinak nyílt forrású implementációin és ezekhez szorosan kapcsolódó projekteken működnek együtt.”

***Java Community Process (JCP)***

1998-ban bevezetett nyílt, a részvételt elősegítő folyamat a Java-hoz kötődő technológiai specifikációk, referencia implementációk és tesztkészletek fejlesztésére és felülvizsgálatára. A JCP program a nemzetközi Java fejlesztői közösséggel együttműködve segíti elő a Java platform fejlődését.

***Java Specification Request (JSR)***

Egy vagy több tag által a JCP-hez benyújtott dokumentum, mely egy új specifikáció kifejlesztését vagy egy létező specifikáció jelentős átdolgozását indítványozza.

***JDK Enhancement Proposal (JEP)***

Egy folyamat a JDK fejlesztésére irányuló javaslatok kezelésére, melyet az OpenJDK projektben használnak. Elsődleges célja javaslatok egy olyan rendszeresen frissített listájának előállítása, mely hosszú távú ütemtervként szolgálhat a JDK kiadásához.

***Oracle JDK és OpenJDK***

**A JDK 11 előtt:**

A két JDK forráskódja nagymértékben azonos, az Oracle JDK azonban tartalmaz nem nyílt forrású üzleti funkciókat is.

**A JDK 11-től:**

Az Oracle JDK és az OpenJDK funkcionálisan azonos és egymással felcserélhető.

A felhasználási feltételek azonban lényegesen eltérőek!

**Java SE aktuális kiadási modellje**

* Fő kiadások félévente
* Frissítések negyedévente
* Hosszú távon támogatott (long term support, LTS) kiadások háromévente

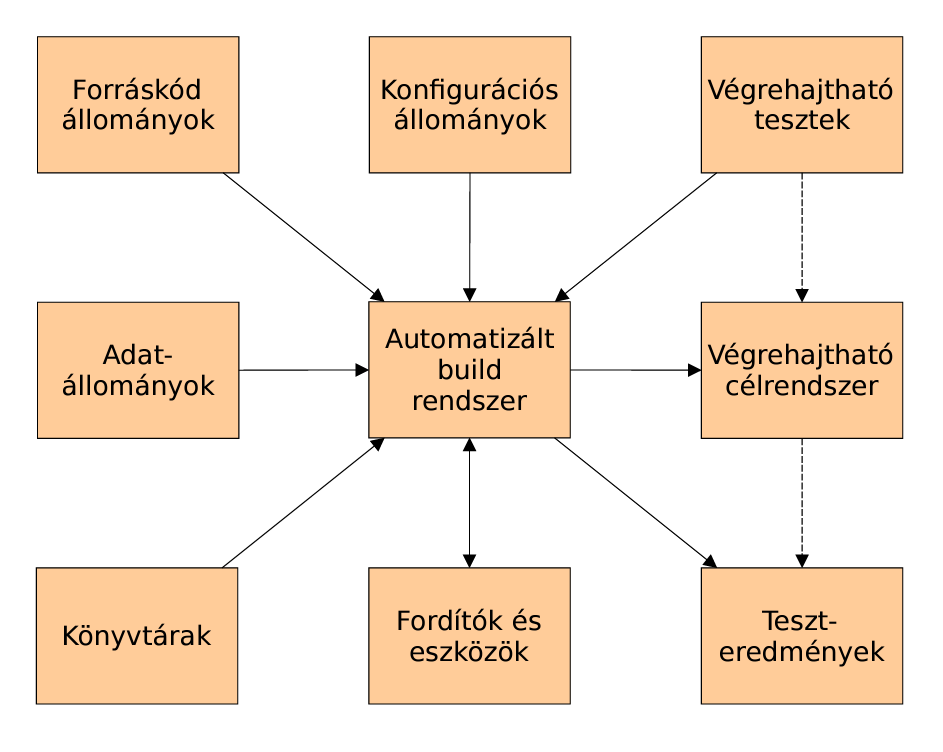
1. Rendszerépítés

**Mi a rendszerépítés?**

A rendszerépítés az a folyamat, melynek során egy teljes végrehajtható rendszer kerül létrehozásra a rendszerkomponensek, külső könyvtárak, konfigurációs állományok és további információk lefordításával és összeszerkesztésével.

**Build automatizálás**

A rendszerépítés nagy mennyiségű, a szoftverrel és a működési környezetével kapcsolatos információ összegyűjtésével jár. Emiatt érdemes egy automatikus build eszközt használni a rendszerépítéshez. Ideális esetben egyetlen paranccsal összeállítható egy teljes rendszer.



**Build eszközök funkciói**

***Build szkript generálás:*** A rendszernek elemeznie kell az összeállítandó programot és automatikusan generálnia kell egy build szkriptet (konfigurációs állományt). A rendszernek támogatni kell a build szkriptek kézi létrehozását és szerkesztését.

***Verziókezelő rendszer integráció:*** A rendszernek le kell tudni szedni a verziókezelő rendszerből a komponensek szükséges verzióit.

***Minimális újrafordítás:*** A rendszernek meg kell határoznia, hogy mely forráskódokat kell újrafordítani és szükség esetén el is kell végeznie a fordítást.

***Végrehajtható rendszer létrehozása:*** A rendszernek össze kell szerkesztenie a lefordított tárgykódú állományokat egymással és más szükséges állományokkal, mint például könyvtárakkal és konfigurációs állományokkal egy végrehajtható rendszer létrehozásához.

***Teszt automatizálás:*** Bizonyos rendszerek automatikusan tudnak futtatni automatikus teszteket teszt automatizálási eszközökkel, mint például a JUnit. Ezek azt ellenőrzik, hogy a változások miatt nem omlik-e össze a build.

***Jelentéskészítés:*** A rendszernek jelentéseket kell adnia az összeállítás sikeréről vagy sikertelenségéről és a lefuttatott tesztekről is.

***Dokumentáció generálás:*** A rendszer képes lehet különféle dokumentációk előállítására, mint például kézikönyv oldalak vagy API dokumentáció.

***Függőségkezelés:*** A rendszer képes lehet a komponensek (például könyvtárak) szükséges verzióit tárolókból letölteni.

**Folyamatos integráció fogalma**

Definíció (SEVOCAB): Módszer, mely a fejlesztett rendszer összeállításához és teszteléséhez egyetlen fővonalba fésüli össze a csapat fejlesztői által végzett forráskód módosításokat.

Definíció (Martin Fowler): A folyamatos integráció egy szofverfejlesztési gyakorlat, melynél egy csapat tagjai gyakran integrálják a munkájukat. Rendszerint mindenki legalább napi rendszerességgel integrál, mely napi többszöri integrációt eredményez. A teszteket is magába foglaló automatikus összeállítás történik minden egyes integráció ellenőrzéséhez, hogy a lehető leghamarabb kerüljenek felfedezésre az integrációs hibák. Sok csapat úgy tapasztalja, hogy ez a megközelítés jelentősen csökkenti az integrációs hibákat és lehetővé teszi a csapat számára erős kohézióval bíró (cohesive) szoftverek gyorsabb kifejlesztését.

**DevOps fogalma**

Definíció (SEVOCAB): Alapelvek és gyakorlatok, melyek jobb kommunikációt és együttműködést tesznek lehetővé az érdekelt felek között szoftver és rendszer termékek és szolgáltatások specifikálása, fejlesztése és működtetése céljából.

1. Apache Maven

**Mi az Apache Maven?**

Egy **projektkezelő eszköz** (software project management and comprehension tool) a következő célkitűzésekkel:

* Az összeállítási folyamat megkönnyítése
* Egységes rendszer biztosítása az összeállításhoz
* Minőségi projekt információk szolgáltatása
* Irányelvek és legjobb gyakorlatok szolgáltatása a fejlesztéshez

A főbb lehetőségek közé tartoznak a következők:

* Egységes rendszer projektek összeállításához
* Függőségkezelés
* Disztribúció közzététel
* Jelentéskészítés és webhely létrehozás

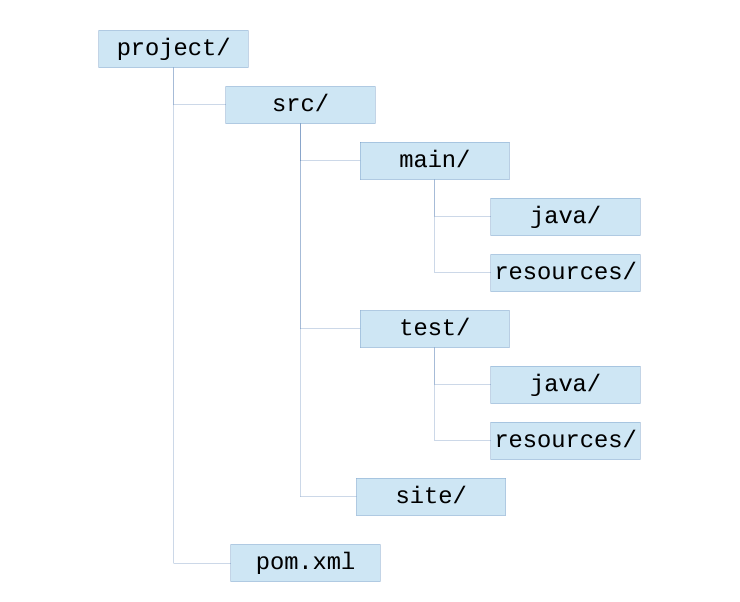
Jellemzők:

* Konvenciók előtérbe helyezése az egyedi beállításokkal szemben (convention over configuration)
* Projekt életciklusok és életciklus fázisok meghatározása
* Jellegét tekintve deklaratív
* Moduláris és kiterjeszthető felépítés (Minden funkció megvalósítása bővítményekkel történik).
* Nem csak Java projektekhez lehet használni (C/C++, Kotlin, Scala).

Fejlesztés:

* Programozási nyelv: Java
* Szabad és nyílt forrású

**A szabványos könyvtárszerkezet**

****

**Parancssori használat**

* A használat módjáról és a megadható parancssori opciókról az mvn --help vagy mvn -h parancsok végrehajtásával kaphatunk leírást.
* Parancssori argumentumként megadható életciklus fázis (például mvn package) és előtag:cél formában bővítmény-cél (például mvn site:run).
* Bővítmény-cél megadható groupId:artifactId:verzió:cél formában is.

**Alapfogalmak**

* **Termék (artifact)**

Egy projekt által előállított állomány, mely annak végső termékének tekinthető.

* **Projekt objektum modell (POM – Project Object Model**)

Egy projekt deklaratív leírását tartalmazó XML dokumentum (pom.xml). Metaadatokat és konfigurációs beállításokat tartalmaz.

* **Szuper-POM (super POM)**

A szuper-POM a Maven által alapértelmezésben használt POM. Ha egy projektnek nincs explicit módon megadott szülője, akkor az úgynevezett szuper-POM beállításait örökli.

* **Effektív POM (effective POM)**

A projekthez tartozó POM, a felmenő ági projektekhez tartozó POM-ok és a szuper-POM kombinációja. A futás során a projekthez ténylegesen felhasználásra kerülő beállításokat szolgáltatja.

* **Maven koordináták (Maven coordinates)**

Minden projektet a Maven koordinátái azonosítanak, mely a következő 3 komponensből áll: groupId, artifactId, version

* **Bővítmény (plugin), bővítmény-cél (plugin goal)**

Szinte minden funkciót bővítmények nyújtanak. A bővítmények egy-egy funkciót megvalósító úgynevezett célokat szolgáltatnak. A bővítmények is termékek, melyekre a Maven koordinátákkal lehet hivatkozni. Minden bővítményhez tartozik egy olyan előtag, mely lehetővé teszi az egyes célokra előtag:cél formában történő hivatkozást.

* **Távoli és lokális tároló (remote/local repository)**

A termékek, köztük a bővítmények elérése tárolókból történik, amelyeknek két fajtája van:

* + A távoli tárolók tipikusan a weben érhetők el, például HTTP vagy HTTPS protokollon keresztül.
  + A lokális tároló a távoli tárolókból a felhasználó számára lokális használatra letöltött termékeket tartalmazza az állományrendszerben, valamint az mvn install paranccsal lokálisan telepített termékeket.

A távoli és lokális tárolók azonos felépítésűek.

* **Életciklus (lifecycle), életciklus fázis (lifecycle phase)**

Egy életciklus jól meghatározott életciklus fázisok egy sorozatát jelenti.

**Életciklusok, életciklus fázisok**

Egy életciklus jól meghatározott életciklus fázisok egy sorozatát jelenti.

* Minden életciklus fázist egy egyedi név azonosít.
* A fázisokhoz bővítmény-célokat lehet hozzárendelni, a hozzárendelést kötésnek nevezik.

Az életciklus fázisok végrehajtása a hozzájuk tartozó bővítmény-célok végrehajtását jelenti.

* Adott fázis végrehajtása maga után vonja valamennyi, a sorrendben azt megelőző fázis végrehajtását.
* Egy fázishoz kötött célok abban a sorrendben kerülnek végrehajtásra, amelyben a POM-ban deklarálására kerülnek.

Három szabványos életciklus: clean, default, site

* A csomagolás módjától függően a fázisokhoz alapértelmezésben hozzárendeltek bizonyos célok.

A **clean** életciklus az alábbi három életciklus fázist tartalmazza:

(1) pre-clean

(2) clean

(3) post-clean

A **site** életciklus az alábbi négy életciklus fázist tartalmazza:

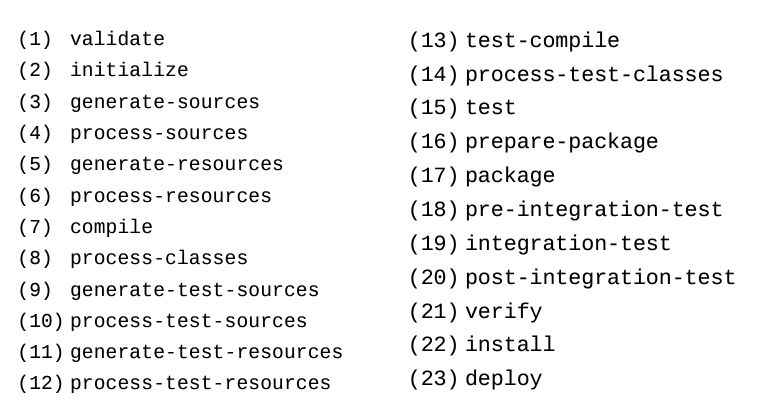
(1) pre-site

(2) site

(3) post-site

(4) site-deploy

A **default** életciklus:



Röviden process-resources -> compile -> process-test-resources -> test-compile -> test -> package -> install -> deploy.

**Tulajdonságok, hivatkozás tulajdonságokra**

A ${x} formájú hivatkozások helyettesítése a POM-ban.

* ${env.név} formájú hivatkozások helyettesítése a megfelelő nevű környezeti változó értékével.
  + Például ${env.PATH} a PATH környezeti változó értékét szolgáltatja.
* A hivatkozásban megadható Java rendszertulajdonság neve.
  + Példa: ${java.home}, ${line.separator}
* ${project.x} formájú hivatkozások helyettesítése a POM megfelelő elemének értékével. Csak egyszerű típusú elemekhez használható!
  + Példa: ${project.groupId}, ${project.artifactId}, $ {project.url}, ${project.build.outputDirectory}
* ${settings.x} formájú hivatkozások helyettesítése a settings.xml állomány megfelelő elemének értékével.

Ilyen módon hivatkozható bármely, a properties elemben definiált tulajdonság.

**Függőségek megadása**

* + **groupId, artifactId, version:** a függőség Maven koordinátáit tartalmazzák
  + **classifier:** az egy projekt által létrehozott termékek megkülönböztetésére szolgál
    - Tipikus értéke például a javadoc és sources.
  + **type:** a függőség típusát tartalmazza (alapértelmezés: jar)
    - A típus meghatározza a termék állománynév kiterjesztését és csomagolását, valamint (opcionálisan) az osztályozót is.
  + **scope:** a függőség hatáskörét tartalmazza, lehetővé teszi a különböző összeállítási folyamatokhoz (például fordítás, tesztelés) szükséges classpath meghatározását és a tranzitivitás korlátozását, lehetséges értékei:
    - ***compile:*** minden classpath tartalmazza a függőséget, a függő projekteknek is függősége lesz (ez az alapértelmezés)
    - ***provided:*** a függőséget a futtató környezet (például a JDK) biztosítja, csak a fordításhoz használt classpath tartalmazza, nem tranzitív
    - ***runtime:*** a függőség csak a végrehajtáshoz szükséges (a programtesztek végrehajtásánál is rendelkezésre áll)
    - ***system:*** a függőséget nem egy tároló szolgáltatja, hanem a lokális állományrendszerben található
    - ***test:*** a függőség csak a programtesztek fordításához és végrehajtásához áll rendelkezésre, nem tranzitív
    - ***import:*** kizárólag pom típusú függőségekhez adható meg a dependencyManagement részben, egy ilyen függőség kicserélésére kerül a POM-ja dependencyManagement részének függőségeire
  + **systemPath:** system hatáskörű függőséghez megengedett és kötelező
    - A függőség abszolút elérési útvonalát tartalmazza, mint például: <systemPath>${java.home}/lib/jfxrt.jar </systemPath>
  + **exclusions:** a kizárandó függőségek megadására szolgál

**Verziószámok**

A verziószámok p.q.r-s alakúak, ahol:

* + - p a főverzió (major version)
    - q az alverzió (minor version)
    - r inkrementális verzió (incremental version)
    - s build szám (build number) vagy minősítő (qualifier)

**Verzió követelmények**

Függőségekben verziószám helyett megadható verziótartomány.

Ha egy függőséghez a version elemben egyetlen verziószám szerepel, akkor a Maven azt csupán ajánlásnak tekinti, melyet szükség esetén tetszőleges verzióval helyettesíthet.

**Tranzitív függőségek, tranzitív függőségek kizárása**

* Ha B függősége A-nak, C pedig B-nek, akkor azt mondjuk, hogy C tranzitív függősége A-nak.
* A Maven automatikusan kezeli a tranzitív függőségeket. Képes a tranzitív függőségek kapcsán felmerülő konfliktusok kezelésére.
* Tranzitív függőségek kizárására szolgál az exclusions elem. Konfliktus esetén szükséges lehet, de hasznos felesleges függőségek kizárásához is.

**Öröklés**

* Olyan projekt lehet szülő, melynél a csomagolás módja (packaging) pom.
* A gyerek projekt a szülő projekthez tartozó POM-ból automatikusan örököl bizonyos beállításokat az effektív POM előállítása során.

**Profilok, profil aktiválás**

A profilok a POM olyan opcionális beállításokat tartalmazó részei, amelyek csak aktiválás esetén kerülnek felhasználásra.

* Lehetővé teszik a POM futásidejű módosítását.
* Hasznosak például a projekt eltérő környezetekben történő használata esetén.

Profil aktiválása történhet a felhasználó explicit kérésére és meghatározott feltételek teljesülése esetén automatikusan.

**Bővítmények használata**

A **plugin** elemben rendelkezésre álló elemek:

* ***groupId, artifactId, version:*** a bővítmény Maven koordinátái
* ***configuration:*** konfigurációs paramétereket tartalmaz a célok végrehajtásához
  + Az XML séma a tartalomra nem tesz semmilyen megszorítást.
  + Ezek a konfigurációs paraméterek valamennyi bővítmény-célra vonatkoznak.
* ***dependencies:*** a bővítményhez szükséges függőségeket tartalmazza
  + A függőségek megadása a korábban tárgyalt formában történik.
* ***executions:*** lehetővé teszi bővítmény-célok végrehajtásának hozzákötését életciklus fázisokhoz, így az összeállítási folyamat testreszabását (részletesen lásd később)
* ***extensions:*** azt jelzi, hogy be kell-e tölteni a bővítmény kiterjesztéseit (alapértelmezés: false)
* ***inherited:*** azt jelzi, hogy öröklés során át kell-e venni a bővítmény beállításait (alapértelmezés: true)

Az **execution** elemben rendelkezésre álló elemek:

* + ***id:*** a végrehajtás egyedi azonosítója
  + ***phase:*** az életciklus fázis neve, melyhez hozzá kell kötni a cél(ok) végrehajtását
  + ***goals/goal:*** a végrehajtandó bővítmény-célok neveit tartalmazzák
  + ***inherited:*** azt jelzi, hogy öröklés során át kell-e venni az execution elemet (alapértelmezés: true)
  + ***configuration:*** konfigurációs paramétereket tartalmaz a goal elemekben felsorolt célok végrehajtásához
    - Általa finomítható a plugin/configuration elemben megadott konfiguráció.

Egy bővítmény-célhoz tartozhat egy alapértelmezett életciklus fázis, ekkor az execution elemben nem szükséges megadni a phase elemet. Ha nincs alapértelmezett életciklus fázis, akkor a phase elem hiányában a bővítmény-cél nem kerül végrehajtásra!

1. Szoftver sebezhetőségek

**Sebezhetőség fogalma**

Egy tökéletlenségből származó hiba egy szoftverben, firmware-ben, hardverben vagy szolgáltatás komponensben, mely kiaknázható az érintett komponens vagy komponensek titkosságára, integritására vagy rendelkezésre állására negatív hatást gyakoroló módon.

**Sebezhetőségi adatbázis fogalma**

Egy sebezhetőségi adatbázis egy speciális adatbázis számítógépes rendszerekben talált sebezhetőségekre vonatkozó információk gyűjtéséhez és kezeléséhez.

**CVE - Common Vulnerabilities and Exposures**

A ***Gyakori Sebezhetőségek és Kitettségek*** program küldetése a nyilvánosságra hozott kiberbiztonsági sebezhetőségek azonosítása, meghatározása és katalogizálása. A katalógusban minden egyes sebezhetőséghez egy CVE rekord van. A sebezhetőségeket a CVE Programmal együttműködő szervezetek fedezik fel és publikálják.

* + - A CVE nem egy sebezhetőségi adatbázis.
    - Egységes azonosítók egy listája közismert kiberbiztonsági sebezhetőségekhez.
    - Lehetővé teszi az információ technológiai és kiberbiztonsági szakértők számára, hogy ugyanarra a problémára hivatkozzanak.
    - Tehát inkább egy szótár, mint adatbázis.

**NVD - National Vulnerability Database**

A **Nemzeti Sebezhetőségi Adatbázis** a NIST által karbantartott, a CVE listára épülő és azzal teljesen szinkronizált sebezhetőségi adatbázis.

* Az NVD a Common Vulnerability Scoring System (CVSS) (Egységes Sebezhetőség Pontozási Rendszer) segítségével rendel egy súlyossági pontszámot minden egyes sebezhetőséghez.
* Az NVD a CVE rekordokat olyan információkkal egészíti ki, mint például a hibajavítási információk vagy súlyossági pontszámok.
* Fejlett keresési lehetőségeket is biztosít, mint például a keresés operációs rendszer szerint vagy a sebezhetőség típus alapján.

1. Szemantikus verziószámozás

**Verziópokol/függőségi pokol**

A ***verziópokol*** egy köznyelvi megnevezés arra, hogy a felhasználó azért nem tud telepíteni egy programot, mivel annak egy másik programnak arra a verziójára van szüksége, amit nem telepíthet, mivel más programok ugyanennek a függőségnek egy másik verzióját használják.

**Mi a szemantikus verziószámozás?**

Az általános bevett gyakorlaton alapuló egyszerű szabályok és követelmények a verziószámok kiosztásához és növeléséhez.

**Normál verziószámok**

A ***normál verziószámok*** X.Y.Z formájúak, ahol X, Y és Z nemnegatív egész számok:

* X: főverzió (major version)
* Y: alverzió (minor version)
* Z: patch verzió.

**Verziószámok növelése**

Növeld

* **a főverziót**, amikor a korábbi verzióval inkompatibilis módon változik az API;
* **az alverziót**, amikor a korábbi verzióval kompatibilis módon vezetünk be új funkcionalitást;
* **a patch verziót**, amikor a korábbi verzióval kompatibilis hibajavítások történnek.

1. Annotációk a Java programozási nyelvben

**Annotáció fogalma**

Egy programkonstrukcióra vonatkozó metaadat, melynek nincs közvetlen hatása a programvégrehajtásra.

* Az annotációk gépi feldolgozásra alkalmasak, fordítási vagy futási időben érhetők el.
* Dokumentációs célokat is szolgálnak: nagyon tömör dokumentációs formának tekinthetők.

**Lehetséges felhasználások**

* ***Információk szolgáltatása a fordítónak:*** például tekintsen el bizonyos figyelmeztetésektől, jelezzen bizonyos hibákat. (@Deprecated, @Override)
* ***Kódgenerálás:*** az annotációk alapján kód generálható.
* ***Futásidejű feldolgozás:*** bizonyos annotációkhoz hozzá lehet férni végrehajtási időben.

**Annotáció szintaxisa**

* Egy annotációt a következők alkotnak:
  + Egy annotáció interfész neve.
  + Opcionálisan egy olyan lista, melyet vesszővel elválasztott elem-érték párok alkotnak.
    - A listát () karakterek között kell megadni.
* Az annotáció interfész határozza meg a használható elem-érték párokat.
  + Nem kötelező az alapértelmezett értékkel rendelkező elemek megadása.
* Az elem-érték párok sorrendje nem lényeges.
  + Az elem-érték párokat abban a sorrendben szokás egy annotációban megadni, melyben az annotáció interfész deklarációjában is deklarálásra kerülnek az elemek.
* Az annotációt az annotáció interfész annotációjának mondjuk.
* Ha egy elem típusa egy tömb típus, akkor az értéket egy tömb inicializáló kifejezés kell, hogy szolgáltassa.

**Annotációk fajtái**

* + Közönséges
  + Egyelemű
  + Jelölő

**Hol alkalmazhatók annotációk?**

***Deklarációkra:***

* Konstruktor, osztályváltozó, enum konstans, lokális változó, metódus, modul, csomag, formális paraméter, osztály, interfész (beleértve az annotáció interfészeket is), enum, típusparaméter, rekord és rekord komponens (Java SE 16) deklarációjára
* Deklaráció annotációknak nevezzük a deklaráció kontextusban előforduló annotációkat.

***Deklarációkban és kifejezésekben használt típusokra (Java SE 8):***

* Típus annotációknak nevezzük a típus kontextusban előforduló annotációkat.

**Előre definiált annotáció interfészek**

* @Deprecated
* @Override
* @SuppressWarnings
* @SafeVarargs
* @FunctionalInterface

**Meta-annotációk**

* @Retention
* @Documented
* @Target
* @Inherited
* @Repeatable

**Annotáció interfészek deklarálása**

Új annotáció interfész létrehozása az alábbi annotáció interfész deklarációval történik:

* módosítók @interface név { deklarációk }

A fenti deklaráció egy speciális interfészt határoz meg.

* A közönséges interfészekre vonatkozó szabályok nem mindegyike vonatkozik az annotáció interfész deklarációkra.

A deklaráció törzsében az alábbi deklarációk megengedettek:

* Osztálydeklaráció
* Interfész deklaráció (annotáció interfész deklaráció is)
* Konstans deklaráció, mint például:
  + int MIN = 0; int MAX = 10;
* Speciális metódus deklaráció

1. A Java SE/SDK új lehetőségei

nem vagyok hajlandó megtanulni :c

1. A Java haladó szintű lehetőségei

**Nem absztrakt (alapértelmezett, statikus, privát) interfész metódusok**

Egy nem absztrakt interfész metódus egy, a default, static vagy private módosítók valamelyikével deklarált interfész metódus, melynek metódustörzse is van.

Implicit módon absztrakt minden olyan interfész metódus, melynek nincs private, default vagy static módosítója.

Az abstract, default és static módosítók kölcsönösen kizárják egymást interfész deklarációknál.

Egy **alapértelmezett metódus** egy interfészben a **default** módosítóval deklarált példánymetódus.

* + A metódustörzs a metódus implementációját szolgáltatja az interfészt a metódus felülírása nélkül implementáló osztályok számára.

Egy **statikus interfész metódus** egy interfészben a **static** módosítóval deklarált metódus.

* + A statikus interfész metódusokat nem öröklik az alinterfészek.
  + A statikus interfész metódusok hívása egy bizonyos példányra történő hivatkozás nélkül történik, az osztályok statikus metódusaihoz hasonlóan.
  + A statikus interfész metódusok lehetővé teszik egy interfészhez kötődő konkrét segédmetódusok hozzáadását közvetlenül magához az interfészhez.

Egy **privát interfész metódus** egy interfészben a **private** módosítóval deklarált metódus.

* A private módosító kombinálható a static módosítóval.
* A privát interfész metódusokat nem öröklik az alinterfészek.
* Az alapértelmezett metódusok és a statikus interfész metódusok közötti kódmegosztásra szolgálnak.

**java.util.Optional**

* ***Egy konténer objektum***, mely vagy tartalmaz egy nem null értéket, vagy nem.
* Elsődlegesen olyan metódusok visszatérési típusaként szolgál, melyeknél egyértelműen szükséges a „nincs eredmény” ábrázolása és ahol null használata valószínűleg hibát okoz.
  + Rákényszeríti a programozót arra, hogy foglalkozzon egy érték hiányával, így tehát segíti a NullPointerException kivételek elkerülését.
* Egy Optional típusú változó értéke soha nem szabad, hogy null legyen, mindig egy Optional példányra kell, hogy mutasson.

**Funkcionális interfészek**

Egy funkcionális interfész egy olyan interfész, melynek **csak egy absztrakt metódusa van**.

* + Egyetlen absztrakt metódusú (Single Abstract Method – SAM) interfészként vagy típusként is ismert.
  + Az egyetlen absztrakt metódust a funkcionális interfész funkcionális metódusának nevezik.
  + Egy funkcionális interfésznek több alapértelmezett, statikus és/vagy privát metódusa is lehet.

**Beépített funkcionális interfészek**

* **java.util.function.Function<T, R>:** Egy eredményt létrehozó egyargumentumú függvényt ábrázol.
* **java.util.function.Predicate<T>:** Egy egyargumentumú predikátumot (logikai értékű függvényt) ábrázol.
* **java.util.function.Supplier<T>:** Egy eredményeket szolgáltató objektumot ábrázol. (nincsenek nem absztrakt metódusai)
* **java.util.function.Consumer<T>:** Egy olyan műveletet ábrázol, mely egyetlen input argumentumot vár és nem ad vissza eredményt.

**Lambda kifejezések**

Egy funkcionális interfészt implementáló névtelen belső osztály egy példányát ábrázolják nagyon tömören.

Egy lambda kifejezés kiértékelése egy funkcionális interfészt implementáló névtelen belső osztály egy példányát hozza létre.

A lambda kifejezések névtelen függvényeket ábrázolnak.

**Metódus referenciák**

Egy metódus referencia arra szolgál, hogy egy metódushívásra hivatkozzunk anélkül, hogy ténylegesen hívás történne.

Egy metódus referencia kifejezés kiértékelése egy funkcionális interfésztípus egy példányát hozza létre.

Fordítási hibát okoz, ha egy metódus referencia kifejezés értékadástól, hívástól vagy típuskényszerítéstől eltérő bármely más szövegkörnyezetben fordul elő egy programban.

**Streamek**

Egy stream elemek egy sorozata, melyen műveletek végezhetők.

***Stream műveletek***

Van **köztes** és van **terminális** művelet.

**Köztes a filter(), map(), sorted()**, **terminális a count(), max(), forEach().**

**A terminális műveletek** (terminal operations) egy streamtől különböző eredményt hoznak létre vagy mellékhatást eredményeznek.

**A köztes műveletek** (intermediate operations) egy új streamet adnak vissza.

**A köztes műveletek további 2 típusba sorolhatóak:**

* Az ***állapotmentes műveletek*** (stateless operations) nem őriznek a korábban látott elemekből állapotot, amikor egy új elemet dolgoznak fel, tehát minden egyes elem a többi elemen történő műveletektől függetlenül dolgozható fel. Példák: filter(), map()
* Az állapotőrző műveletek (stateful operations) felhasználhatnak a korábban látott elemekből állapotot, amikor új elemeket dolgoznak fel, eredmény létrehozása előtt szükséges lehet számukra a teljes bemenet feldolgozása. Példák: distinct(), sorted()

**Bizonyos műveletek rövidzár műveletnek (short-circuiting operation) vannak jelölve:**

* Egy köztes művelet rövidzár, ha végtelen bemenetből is létrehozhat eredményként egy véges streamet. Példák: limit (), takeWhile ()
* Egy terminális művelet rövidzár, ha végtelen bemenet esetén is befejezheti működését véges időben. Példák: anyMatch (), findFirst()

***Viselkedési paraméterek kívánatos jellemzői***

* **Interferencia-mentesség (non-interference):** A viselkedési paraméterek nem szabad, hogy módosítsák a stream adatforrását.
* **Állapotmentesség (statelessness):** Egy lambda kifejezés (vagy a megfelelő funkcionális interfészt implementáló más objektum) állapotőrző, ha eredménye olyan állapottól függ, mely a stream csővezeték végrehajtása során megváltozhat. A stream csővezeték eredménye nemdeterminisztikus vagy rossz lehet, ha a stream műveletek viselkedési paraméterei állapotőrzők.

***Stream csővezeték, műveletek kiértékelése, csővezeték végrehajtás***

Stream műveletek egy **csővezetékké** láncolhatók össze. Egy stream csővezeték egy forrásból áll, melyet nulla vagy több köztes művelet és egy terminális művelet követ.

* A köztes műveletek mindig **lusta** **kiértékelésűek**.
  + Egy köztes művelet végrehajtása ténylegesen nem eredményez semmiféle műveletvégzést.
  + A csővezeték forrásának bejárása nem kezdődik el, míg a terminális művelet végrehajtásra nem kerül.
* Majdnem minden esetben **mohó** **kiértékelésűek** a terminális műveletek.
  + Egy terminális művelet végrehajtása indítja el az adatforrás bejárását, a csővezeték feldolgozása a visszatérés előtt fejeződik be.

A terminális művelet **végrehajtása** során a csővezeték elhasználódik és többé nem használható.

***Redukciós műveletek***

Egy redukciós művelet egy **terminális** művelet, mely egy input elemsorozatból egyetlen összesítő eredményt képez egy egyesítő művelet ismételt alkalmazásával. Példák: számok összegének vagy maximumának meghatározása, elemek összegyűjtése egy listába.

**Reduce:** A reduce() egy terminális művelet, mely elemek egy sorozatát egyetlen elemre redukálja.

**Collect:** Egy módosítható redukciós művelet egy módosítható eredmény konténerbe (mint például egy kollekció vagy egy StringBuilder) gyűjti össze az input elemeket a stream elemeinek feldolgozásakor.

***Párhuzamosság***

Minden stream művelet végrehajtható szekvenciálisan és párhuzamosan is.

Az alábbi módszerek valamelyikével kapható egy párhuzamos stream:

* + Egy kollekció parallelStream() metódusának meghívásával.
  + Egy már létező szekvenciális stream parallel() metódusának meghívásával.

1. Git

**git init:** új tároló létrehozása

**.gitignore:** állományok figyelmen kívül hagyása

**git status:** munkakönyvtár állapotának vizsgálata

**git diff:** nem véglegesített módosítások megjelenítése

**git commit:** változtatások véglegesítése

**git add:** állományok hozzáadása tárolóhoz

**git rm:** állományok törlése tárolóból

**git log:** verziótörténet megjelenítése

**ágak használata:**

* branch: kilistázza a tároló lokális ágait
* branch [név]: létrehoz egy ágat adott néven
* checkout: egyik ágról a másikra lehet váltani vele
* merge: adott ágat másol a jelenlegi ágba

**módosítások „visszacsinálása”:**

* revert: ha már commitoltad a faszságot
* reset: ha még nem commitoltad a faszságot
* amend: utolsó commit üzenet megváltoztatása