Ant, *Gradel* i *Maven*, usporedba aplikacijske logike *Java Swing* i *JavaFX*

Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

11. siječnja, 2023.

Alati za izgradnju softvera - *Apache Ant*

Apache Ant je *Java* **biblioteka i alat komandne linije** čiji cilj je **upravljati zadacima opisanim u dokumentima kreiranja** (eng. *build files*) i **međuovisnim ekstenzijama**. *Ant* se najviše koristi kao alat za izgradnju *Java* aplikacija. On može obavljati nekoliko zadataka koji omogućuju **prevođenje**, **sastavljanje**, **testiranje** i **pokretanje** *Java* aplikacija. Može se koristiti i za stvaranje aplikacija drugih programskih jezika (npr. $C/C++$). Također, može upravljati proizvoljnim procesom koji uključuje **ciljeve i zadatke**. *Ant* je napisan u *Javi*, **dozvoljava pisanje dokumenata koji sadrže zadatke i tipove** (slično bibliotekama), ne obvezuje na korištenje konvencija kodiranja ili izgled direktorija *Java* projekta koji ga koristi.

Dokumenti kreiranja su `XML` dokumenti koji **sadrže jedan projekt**. Projekt sadrži tri ne obavezna atributa: **ime** (eng. *name*) - definira ime projekta, **default** - definira standardni cilj (ukoliko niti jedan drugi nije definiran), **osnovni direktorij** (eng. *basedir*) - polazni direktorij od kojeg se računaju relativne putanje.

Alati za izgradnju softvera - *Apache Ant*

Ukoliko nije definiran, relativne putanje se računaju od direktorija koji sadrži dokument kreiranja. **Opis projekta** se može zadati kao `<description>` element najvišeg nivoa.

Svaki projekt definira **najmanje jedan cilj**. Ciljevi sadrže **elemente zadatka** koji mogu imati **atribut** `id` s jedinstvenom vrijednosti (može se koristiti za referiranje na zadatak). Kod pokretanja *Ant*-a, korisnik može definirati koje zadatke želi izvršiti (inače se izvršava standardni zadatak - *default*).

Cilj može ovisiti o drugim ciljevima a *Ant* **može razriješiti te ovisnosti preko atributa** `depends` (prvo se obrade ovisnosti a zatim cilj). Npr. možemo imati ciljeve za *prevođenje*, ciljeve za *stvaranje izvršne datoteke* itd. (cilj za stvaranje izvršne datoteke ovisi o cilju za prevođenje).

Atribut `depends` ne garantira da će se neki cilj izvršiti (npr. ukoliko se cilj koji ga specificira kao ovisnost ne izvrši).

Alati za izgradnju softvera - *Apache Ant*

Zadatak je dio koda koji se može izvršiti, **može imati više atributa** (argumenata). Vrijednost atributa može sadržavati reference na svojstva, a te reference se izvrijednjavaju prije izvršavanja zadatka. Zadaci se zadaju u formatu `<imeZadatka atribut1="vrijednost1" atribut2="vrijednost2" ... />`. Zadacima se može dodijeliti id definiranjem <imeZadatka id="ID" ... />, a taj ID se može koristiti za **dohvaćanje zadatka i dodijeljivanje vrijednosti atributa**.

Postoji niz **predefiniranih** zadataka kao npr. `Ant` - pokrene *Ant* na zadanom dokumentu kreiranja, `Chmod` - promijeni dopuštenja pristupa direktoriju ili dokumentu, `Copy` - kopiraj datoteku ili grupu resursa u drugu datoteku ili direktorij, `Depend` - definira ovisnosti, `Echo` - proslijedi poruku slušačima i mehanizmima za logiranje poruka, `GZip` - pakira resurs koristeći *GZip*, `Import` - uključi neki drugi dokument kreiranja u trenutni projekt, `Jar` - komprimira skupinu dokumenata u `.jar` datoteku, `Java` - izvršava *Java* klasu unutar *Java Virtualnog Stroja* od *Apache Ant*-a ili unutar novo kreirane instance *Java Virtualnog Stroja*.

Alati za izgradnju softvera - *Apache Ant*

`Javac` - prevodi stablo izvornih *Java* datoteka, `Javadoc` - generira dokumentaciju koristeći alat `javadoc`, `Javah` - kreira *JNI* zaglavlje iz izvornog koda *Java* klase, `JUnit` - pokreće testove iz skupa alata za testiranje `JUnit`, `LoadFile` - učitava datoteke, `Mkdir` - kreira direktorij, `Record` - osluškivač trenutnog procesa izgradnje koji zapisuje izlaz u datoteku, `Script` - izvršava skriptu u podržanom jeziku, `Sql` - izvršava niz *SQL* upita koristeći `JDBC`, `Typedef` - dodaje zadatak ili definiciju tipa podataka trenutnom projektu, `Unzip` - vrši akciju ekstrakcije datoteka iz `.zip`, `.war` ili `.jar` datoteke, `XmlValidate` - provjeri ispravnost `XML` dokumenta, `Zip` - kreira `.zip` datoteku i mnogi drugi[1].

Međutim, korisnik može **stvoriti** i **svoje zadatke** definiranjem *Java* klase koja **nasljeđuje** `org.apache.tools.ant.{Task, AbstractCvsTask, JDBCTask, MatchingTask, Pack, Unpack, DispatchTask }`.

[1] Cijela lista, opisi i primjeri predefiniranih zadataka se mogu vidjeti u dokumentaciji *Ant*-a (`https://downloads.apache.org/ant/manual/`)

Alati za izgradnju softvera - *Apache Ant*

Za svaki atribut zadatka, klasa treba sadržavati metodu `setter`, `public void` metodu koja prima jedan argument. Ime metode počinje sa `set`, nakon čega slijedi ime atributa (prvo slovo veliko, ostala malo). Npr. za postavljanje atributa `file` odgovarajuća metoda se zove `setFile`.

Ukoliko zadatak sadrži druge zadatke kao ugniježdene elemente, tada klasa mora implementirati sučelje `org.apache.tools.ant.TaskContainer` (tada zadatak ne može podržavati nikakve druge ugniježdene elemente).

Ukoliko bi zadatak trebao podržavati znakovne podatke, treba implementirati metodu `addText(String)`.

Za svaki ugniježdeni element treba implementirati `create`, `add` ili `addConfigured` metodu. `create` mora biti `public` bez parametara i mora vraćati tip `Object`. Ime metode mora početi s `create` nakon čega slijedi ime elementa. Metoda `add` ili `addConfigured` mora biti `public void` i primati jedan parametar tipa `Object` koji ima konstruktor bez parametara. Ime metoda `add` (`addConfigured`) mora početi s `add` (`addConfigured`) nakon čega slijedi ime elementa.

Alati za izgradnju softvera - *Apache Ant*

Završni korak je implementacija funkcije `public void execute()` koja prijavljuje `BuildException`. Ta funkcija implementira zadatak.

Životni ciklus zadatka:

1. `XML` element koji sadrži oznaku koja odgovara zadatku se pretvara u element tipa `UnknownElement` tijekom parsiranja. `UnknownElement` se stavlja u listu unutar ciljnog objekta ili rekurzivno unutar nekog drugog elementa tipa `UnknownElement`.
2. Kada se izvršava cilj, svaki `UnknownElement` se poziva korištenjem metode `perform()` (time se pokreće zadatak).
3. Zadatak dobiva reference na svoj projekt i lokaciju unutar dokumenta kreiranja kroz naslijeđeni projekt i lokacijske varijable.
4. Ukoliko korisnik definira atribut `id` zadatku, projekt prilikom izvođenja registrira referencu na taj zadatak.
5. Zadatak dobiva referencu na cilj kojem pripada kroz naslijeđenu ciljnu varijablu.
6. Za vrijeme izvođenja se poziva metoda `init()`.

Životni ciklus zadatka:

7. Svi elementi djeca `XML` elementa koji odgovara zadatku su kreirani metodom `createXXX()` tog zadatka ili su instancirani i dodani zadatku metodom `addXXX()` prilikom izvođenja. Elementi djeca koji su dodani za metodu `addConfiguredXXX()` se stvaraju ali se ne poziva stvarna metoda `addConfigured`.
8. Svi atributi zadatka se postavljaju odgovarajućim `setXXX()` metodama pri izvođenju.
9. Sadržaj znakovnih podataka unutar `XML` elementa koji odgovara ovom zadatku se dodaje zadatku metodom `addText()` pri izvođenju.
10. Svi atributi elemenata djece se postavljaju koristeći njihove `setXXX()` metode pri izvođenju.
11. Ukoliko su elementi djeca `XML` elementa koji odgovara zadatku stvoreni za metodu `addConfiguredXXX()` tada se te metode pozivaju u ovom trenutku.
12. `execute()` se poziva tijekom izvođenja. Ukolio `target1` i `target2` oba ovise o `target3`, tada naredba `ant target1 target2` pokreće zadatak `target3` dva puta.

Alati za izgradnju softvera - *Apache Ant*

```java
package mojpaket;

import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class NoviZadatak extends Task {
    private String p;

   //metoda koja izvodi zadatak
    public void execute() throws BuildException {
        System.out.println(p);
    }

    //setter za atribut "poruka"
    public void setPoruka(String p) {
        this.p = p;
    }
}
```

Primjer definiranja zadatka.

Alati za izgradnju softvera - *Apache Ant*

- Klasa koja implementira zadatak mora biti u putanji (`classpath`) kod pokretanja *Ant*-a.
- Projektu treba dodati `<taskdef>` element. To dodaje zadatak sustavu.
- Zadatak se može koristiti u ostatku dokumenta kreiranja.

```
1 <?xml version="1.0"?>
2
3 <project name="PrimjerZadatka" default="main" basedir=".">
4   <taskdef name="zadatak" classname="mojpaket.NoviZadatak"/>
5
6   <target name="main">
7     <zadatak poruka="Pozdrav! Definirani zadatak!"/>
8   </target>
9 </project>
```

Primjer definiranja zadatka.

Alati za izgradnju softvera - *Apache Ant*

```
<?xml version="1.0"?>

<project name="PrimjerZadatka2" default="main" basedir=".">

  <target name="build" >
    <mkdir dir="build"/>
    <javac srcdir="source" destdir="build"/>
  </target>

  <target name="declare" depends="build">
    <taskdef name="zadatak"
        classname="mojpaket.NoviZadatak"
        classpath="build"/>
  </target>

  <target name="main" depends="declare">
    <zadatak poruka="Pozdrav! Definirani zadatak!"/>
  </target>
</project>
```

Definicija zadatka unutar dokumenta kreiranja koji vrši prevođenje.

Alati za izgradnju softvera - *Apache Ant*

Zadatak se može koristiti i **direktno** iz dokumenta kreiranja koji ga je kreirao. U tom slučaju treba **postaviti deklaraciju** `<taskdef>` unutar cilja nakon dijela za prevođenje. `classpath` atribut od `<taskdef>` treba **pokazivati na kod koji je upravo preveden**.

Ant može generirati **događaje** dok izvodi zadatke potrebne za izgradnju projekta. **Slušači** se mogu definirati u *Ant*-u da bi se dohvatili ti događaji. Takvi događaji i slušači se mogu koristiti npr. kod integracije *Ant*-a s **grafičkim sučeljem** ili **razvojnim okruženjem**.

```
1 public class LogAdapter implements BuildListener {
2
3     private MojLogger getLogger() {
4         final MojLogger log = MojLoggerFactory.getLogger(
    Project.class.getName());
5         return log; }
```

Definicija adaptera za logiranje događaja.

Alati za izgradnju softvera - *Apache Ant*

```
1     @Override public void buildStarted(final BuildEvent
      event) {
2         final MojLogger log = getLogger();
3         log.info("Build started."); }
4
5     @Override public void buildFinished(final BuildEvent
      event) {
6         final MojLogger logger = getLogger();
7         MojLogLevelEnum loglevel = ... // mapira event.
      getPriority() u enum koristeci konstante Project.MSG_*
8         boolean OK = event.getException() == null;
9         String logporuka = ... // kreiraj log poruku
      koristeci podatke o dogadaju i pozvanu poruku
10        logger.log(loglevel, logporuka);
11    }
12
13    // implementacija preostalih metoda
14 }
```

Definicija adaptera za logiranje događaja.

Alati za izgradnju softvera - *Apache Ant*

Svojstva (eng. *properties*) se koriste za **uštimavanje procesa stvaranja** ili za **definiranje pokrata stringova koji se koriste unutar dokumenta stvaranja**. U najjednostavnijem obliku, svojstva se definiraju unutar dokumenta kreiranja, međutim mogu se postaviti i izvan okruženja *Ant*. Svojstvo ima **ime** (razlikujemo mala i velika slova) i **vrijednost**. Svojstva se mogu koristiti kao **vrijednosti atributa** zadataka ili **unutar ugniježdenog teksta zadataka** koji ih podržavaju. To se radi navođenjem svojstva između "${" i "}" u vrijednosti atributa. Npr. ukoliko postoji svojstvo `builddir` s vrijednosti `build`, tada se ona može koristiti kao atribut navođenjem ${builddir}/classes. Kod izvođenja se to prevodi u `build/classes`. Kasnije verzije *Ant*-a dozvoljavaju **ekstenzije** svojstva koje mogu uključiti ili isključiti neki element.

Alati za izgradnju softvera - *Apache Ant*

```
1  <project name="MojProjekt" default="dist" basedir=".">
2    <description>
3      jednostavan dokument kreiranja
4    </description>
5    <!-- postavljamo globalna svojstva dokumenta stvaranja -->
6    <property name="src" location="src"/>
7    <property name="build" location="build"/>
8    <property name="dist" location="dist"/>
9
10   <target name="init">
11     <!-- kreiramo vremenski zig -->
12     <tstamp/>
13     <!-- kreiramo strukturu direktorija za izgradnju koju
       koristi compile -->
14     <mkdir dir="${build}"/>
15   </target>
```

Primjer dokumenta kreiranja.

Alati za izgradnju softvera - *Apache Ant*

```
<target name="compile" depends="init"
        description="prevedi izvorne datoteke">
  <!-- Prevedi Java kod iz ${src} u ${build} -->
  <javac srcdir="${src}" destdir="${build}"/>
</target>

<target name="dist" depends="compile"
        description="generiraj dokumente za distribuciju">
  <!-- Kreiraj direktorij za distribuciju -->
  <mkdir dir="${dist}/lib"/>

  <!-- Stavi sve iz ${build} u MyProject-${DSTAMP}.jar
  dokument -->
  <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar"
  basedir="${build}"/>
</target>
```

Primjer dokumenta kreiranja.

Alati za izgradnju softvera - *Apache Ant*

```
  <target name="clean"
        description="ocisti">
    <!-- Izbrisi ${build} i ${dist} direktorije -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

Primjer dokumenta kreiranja.

Projekt može imati niz **tokena** koji se obrađuju prilikom filtriranja kod kopiranja dokumenata.

```
<filter token="godina" value="2021"/>
<copy todir="${dest.dir}" filtering="true">
  <fileset dir="${src.dir}"/>
</copy>

```

Definiranje tokena.

Svako pojavljivanje stringa @godina@ se mijenja stringom "2021".

Alati za izgradnju softvera - *Apache Ant*

Moguće je definirati `PATH` i `CLASSPATH` vrste referenci. Atribut `location` definira relativnu putanju **jednog dokumenta** ili **direktorija** u odnosu na bazni direktorij projekta dok atribut `path` prima listu (separiranu zarezom ili točka-zarezom) lokacija. `path` bi se trebao koristiti s predefiniranim skupom putanja.

```
1 <classpath>
2   <pathelement path="${classpath}"/>
3   <pathelement location="lib/helper.jar"/>
4 </classpath>
5
```

Primjer definicije putanje.

Elementi projekta kojima je definiran atribut `id` se mogu referencirati korištenjem atributa `refid`.

Unutar okruženja *Apache Netbeans* možemo stvoriti aplikaciju koja koristi *Ant* za kreiranje izvršne datoteke (tzv. *Java with Ant* projekt).

Alati za izgradnju softvera - *Gradle*

Gradle je **alat za automatizaciju izgradnje softvera** (eng. *build*). Glavna prednost mu je **brzina izgradnje** softvera, ima dobru pokrivenost unutar alata, razvojnih okruženja i softvera te **mogućnost integracije** s ostalim alatima za razvoj softvera (*Ant*, *Maven*). Nudi brojne mogućnosti kao što su: a) **inkrementalna izgradnja** - provjerava je li se implementacija zadatka promijenila od zadnjeg poziva izgradnje, ukoliko nema promjene zadatak ponovne izgradnje se ne izvodi, b) **cache-iranje izgradnje** - zadatak koji je izvršen na drugom računalu se može lokalno preskočiti i samo učitati iz dijeljene *cache* memorije (funkcionira i lokalno), c) **inkrementalni podzadaci** - zadaci koji su se promijenili se ponovo izvršavaju, međutim ukoliko se neki dokumenti nisu promijenili ne trebaju se nužno ponovo izgrađivati, d) **inkrementalna obrada anotacija** - povećava efikasnost inkrementalnog prevođenja, e) omogućava korištenje višedretvenog prevođenja stvaranjem **demona prevodioca**, f) omogućava **paralelno izvođenje zadataka**, g) **paralelno skidanje podataka i paketa** (tzv. artefakata) potrebnih za izgradnju softvera, h) **ograničenje vremena izvršavanja zadataka**.

Alati za izgradnju softvera - *Gradle*

Gradle omogućava **kontinuiranu izgradnju** (prati sve promjene ulaza zadataka i automatski izvršava zadatak ukoliko dođe do promjene). Zadaci se reprezentiraju u obliku **usmjerenog acikličnog grafa**. Omogućava **paralelno razvijanje projekta i ovisnih biblioteka**, **isključivanje nekih zadataka**, **ispitivanje zadataka** koji bi se izvršili bez izvođenja izvršavanja, izvršava što je više moguće zadataka **unatoč greškama** da bi se otkrila što veća količina grešaka pri jednom pokretanju. Brine se za dohvaćanje i razrješavanje tranzitivnih ovisnosti. *Gradle* podržava *Javu*, *Scalu*, *C/C++* s mnogim prevodiocima te izgradnju softvera na platformi *Android*.

Gradle skripte za izgradnju softvera se zapisuju koristeći *Groovy*[2] `API` ili *Kotlin* [3] `API`, međutim te skripte mogu koristiti i standardni *Java* `API`.

Gradle koristi posebnu dretvu koja se zove **Gradle demon** (eng. Gradle deamon) - **dugotrajni process** koji se izvodi u pozadini. On osigurava da se *Java Virtualni stroj* pokreće **samo jednom** tijekom korišenja *Gradle*-a.

[2] https://groovy-lang.org/
[3] https://kotlinlang.org/

Alati za izgradnju softvera - *Gradle*

Gradle demon osigurava **pamćenje** informacija o strukturi projekta, dokumentima, zadacima te time omogućava brojne ranije navedene optimizacije **inkrementalne obrade** i **kontinuirane izgradnje**.

Svaka aplikacija izgrađena *Gradle*-om se sastoji od jednog ili više **projekata** (npr. `JAR` koji sadrži *Java* biblioteku, web aplikacija itd). Projekt može biti i **niz koraka** koje treba izvršiti (npr. postavljanje aplikacije u produkcijsko okruženje). Posao koji *Gradle* treba napraviti se definira kroz **zadatke** - **atomarne jedinice posla** koje se izvode tijekom izgradnje softvera (npr. prevođenje klasa, kreiranje `JAR` dokumenta, *Java* dokumentacije itd.). Tipično se zadaci enkapsuliraju u **dodatak** (eng. plugin) koji se može uključiti na razna mjesta i u razne procese izgradnje softvera.

Alati za izgradnju softvera - *Gradle*

```
tasks.register('pozdrav') {
  doLast {
  println 'Pozdrav!'
  }
}

```

Skripta za izgradnju softvera u Gradle-u koristeći Groovy (build.gradle).

```
tasks.register("pozdrav") {
  doLast {
  println("Pozdrav!")
  }
}

```

Skripta za izgradnju softvera u Gradle-u koristeći Kotlin (build.gradle.kts).

Pokretanje skripte naredbom `gradle -q pozdrav` ispisuje `Pozdrav!`

Alati za izgradnju softvera - *Gradle*

Gradle skripte koje sadrže zadatke mogu sadržavati programski kod:

```
tasks.register('broji') {
  doLast {
  4.times { print "$it " }
  }
}

```

Skripta za izgradnju softvera u Gradle-u koristeći Groovy (build.gradle).

```
tasks.register("broji") {
  doLast {
  repeat(4) { print("$it ") }
  }
}

```

Skripta za izgradnju softvera u Gradle-u koristeći Kotlin (build.gradle.kts).

Izlaz `gradle -q broji` je: 0 1 2 3.

Alati za izgradnju softvera - *Gradle*

Gradle zadaci mogu ovisiti o drugim zadacima:

```
1  tasks.register('pozdrav,') {
2    doLast {
3    println 'Pozdrav!'
4    }
5  }
6  tasks.register('uvod') {
7    dependsOn tasks.pozdrav,
8    doLast {
9    println "Ja sam Gradle"
10   }
11 }
12
```

Skripta za izgradnju softvera u Gradle-u koristeći Groovy (build.gradle).

Alati za izgradnju softvera - *Gradle*

```
1  tasks.register("pozdrav") {
2    doLast {
3    println("Pozdrav!")
4    }
5  }
6  tasks.register("uvod") {
7    dependsOn("pozdrav")
8    doLast {
9    println("Ja sam Gradle")
10   }
11 }
12
```

Skripta za izgradnju softvera u Gradle-u koristeći Kotlin (build.gradle.kts).

Izlaz `gradle -q uvod` je:

```
Pozdrav!
Ja sam Gradle
```

Alati za izgradnju softvera - *Gradle*

Zadaci u *Gradle*-u se mogu registrirati i programski, mogu se dodavati funkcionalnosti zadacima nakon registracije, koristiti *Ant* zadaci, metode, vanjski resursi itd.

```
1 buildscript {
2   repositories {
3   mavenCentral()
4   }
5   dependencies {
6   classpath group: 'commons-codec', name: 'commons-codec',
      version: '1.2'
7   }
8 }
9
```

Korištenje vanjskih resursa u Gradle-u koristeći Groovy (build.gradle).

Alati za izgradnju softvera - *Gradle*

```
buildscript {
  repositories {
  mavenCentral()
  }
  dependencies {
  "classpath"(group = "commons-codec", name = "commons-codec
    ", version= "1.2")
  }
}

```

Korištenje vanjskih resursa u Gradle-u koristeći Kotlin (build.gradle.kts).

Alati za izgradnju softvera - *Gradle*

Inicijalizacijske skripte se izvode prije izgradnje softvera u *Gradle*-u.

```
allprojects {
  repositories {
  mavenLocal()
  }
}

```

Inicijalizacijska skripta u Gradle-u koristeći Groovy (init.gradle).

```
repositories {
  mavenCentral()
}
tasks.register('pokaziRep') {
  doLast {
  println "Svi rep:"
  println repositories.collect { it.name }
  }
}

```

Skripta za izgradnju softvera u Gradle-u koristeći Groovy (build.gradle).

Alati za izgradnju softvera - *Gradle*

```
allprojects {
  repositories {
  mavenLocal()
  }
}

```

Inicijalizacijska skripta u Gradle-u koristeći Kotlin (init.gradle.kts).

```
repositories {
  mavenCentral()
}
tasks.register("pokaziRep") {
  doLast {
  println("Svi rep:")
  println(repositories.map { it.name })
  }
}

```

Skripta za izgradnju softvera u Gradle-u koristeći Kotlin (build.gradle.kts).

Alati za izgradnju softvera - *Gradle*

`gradle -init-script init.gradle -q pokaziRep` će ispisati:

```
Svi rep:
[MavenLocal, MavenRepo]
```

Gradle omogućava izgradnju softvera koji koristi **nekoliko pod-projekata**:

```
.
|-- app
|  ...
|  |-- build.gradle
|-- lib
|  ...
|  |-- build.gradle
|-- settings.gradle
```

Alati za izgradnju softvera - *Gradle*

Navedeni složeni softver se konstruira kreiranjem dokumenta `settings.gradle`:

```
rootProject.name = 'osnovni-viseprojektni'
include 'app'
include 'lib'
```

settings.gradle.

```
rootProject.name = "osnovni-viseprojektni"
include("app")
include("lib")
```

settings.gradle.kts.

Dodavanje dodataka u skriptu za izgradnju softvera se vrši:

```
plugins {
  id 'com.jfrog.bintray' version '1.8.5'
}
```

build.gradle.

Alati za izgradnju softvera - *Gradle*

```
1 plugins {
2   id("com.jfrog.bintray") version "1.8.5"
3 }
```

build.gradle.kts.

Gradle izgradnja se sastoji od tri glavne faze: a) **inicijalizacija** - određuje se koji projekti su dio izgradnje, kreiraju se instance projekta za svakog od njih, b) **konfiguracija** - tijekom te faze se konfiguriraju objekti projekta i izvršavaju se skripte stvaranja svih projekata koji su dio izgradnje, c) **izvršavanje** - određuje se podskup zadataka, od kreiranih i konfiguriranih tijekom faze konfiguriranja, koji se trebaju izvršiti te se ti zadaci izvršavaju.

Unutar okruženja *Apache Netbeans* možemo stvoriti aplikaciju koja koristi *Gradle* za kreiranje izvršne datoteke (tzv. *Java with Gradle* projekt).

Apache Maven

Apache Maven je alat za **razumijevanje i upravljanje projektima**. **Baziran je na konceptu modela projektnog objekta** (`POM`) te podržava **upravljanje: a) izgradnjom projekta, b) dojavljivanjem i c) dokumentacijom** iz centralnog repozitorija. Glavni ciljevi alata su: a) **olakšati proces izgradnje projekta** - osigurava da se razvojni programer ne mora zamarati svim detaljima izgradnje (izgradnja znatno sporija u odnosu na *Gradle*), b) **pružanje unificirnog sustava izgradnje** - gradi projekt koristeći model projektnog modela i skup dodataka, c) **pružanje kvalitetnih informacija o projektu** (npr. zapis promjena generiran iz izvornog koda, povezivanje različitih izvora, mailing liste održavane od strane projekta, ovisnosti projekta, izvješća o jediničnim testovima i njihovoj pokrivenosti), d) poticanje na **korištenje dobre prakse razvoja softvera** (razvojni ciklus sadrži specifikaciju, izvršavanje i izvještavanje o jediničnim testovima).

Apache Maven

Korisne prakse kod testiranja uključuju **održavanje izvornog koda za testiranje u posebnom ali paralelnom stablu izvornih datoteka**, **korištenje konvencije imenovanja testova za njihovo lociranje i izvršavanje**, **posebno uštimavanje okruženja od strane testova** umjesto posebne izgradnje softvera u svrhu testiranja, **potpora kod stvaranja konačne verzije softvera i obrade potencijalnih problema**. *Maven* ima dosta **stroge smjernice oko strukture direktorija projekta** što ponekad može biti i mana (u nekim ekstremnim slučajevima može dovesti do potpune nekompatibilnosti projekta s alatom).

Apache Maven koristi **nacrt**, standardnu strukturu direktorija koja se uz dodatni ulaz korisnika modificira za potrebe projekta. Ključni dokument koji definira zadatke *Maven*-a se zove **Project Object Model** (POM) i unutar projekta se kreira kao datoteka `pom.xml`.

Apache Maven

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi
    ="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>my-app</name>
  <!-- Promijeniti u web stranicu projekta -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.
    sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>
```

Primjer dokumenta pom.xml.

Apache Maven

```
1   <dependencies>
2     <dependency>
3       <groupId>junit</groupId>
4       <artifactId>junit</artifactId>
5       <version>4.11</version>
6       <scope>test</scope>
7     </dependency>
8   </dependencies>
9
10  <build>
11    <pluginManagement><!-- zapisati verzije dodataka da se
      ne koriste standardne vrijednosti -->
12        ... lista dodataka
13    </pluginManagement>
14  </build>
15 </project>
```

Primjer dokumenta pom.xml.

Apache Maven

- **project** - element najvišeg nivoa u svim *Maven* `pom.xml` dokumentima.
- **modelVersion** - definira verziju objektnog modela koje koristi ovaj `POM`. Osigurava stabilnost ukoliko je potrebno promijeniti model.
- **groupId** - jedinstveni identifikator organizacije ili grupe koja je stvorila projekt. Obično se bazira na punom domenskom imenu organizacije (npr. `org.apache.maven.plugins`).
- **artifactId** - jedinstveno ime primarnog artefakta (uobičajeno `.jar` dokumenta) koji projekt generira. Sekundarni artefakti (skupovi izvornih datoteka) također koriste `artifactId` kao dio svog konačnog imena. Tipični *Maven* artefakt ima oblik `<artifactId>-<version>.<extension>` (npr. `myapp-1.0.jar`).
- **version** - označava verziju artefakta generiranog od strane projekta. *Maven* koristi identifikator `SNAPSHOT` ukoliko je projekt u stanju razvoja.
- **name** - označava ime projekta za prikaz (npr. u dokumentaciji).

Apache Maven

- **url** - definira poveznicu na web stranicu projekta. Često se koristi u generiranoj dokumentaciji projekta.
- **properties** - sadrži vrijednosti varijabli koje su dostupne na proizvoljnom mjestu unutar `POM`-a.
- **dependencies** - djeca ovog elementa sadrže listu ovisnosti (ključan element `POM`-a).
- **build** - sadrži deklaraciju strukture direktorija projekta i upravlja dodacima.

Maven projekt se prevodi naredbom `mvn compile`, a testovi se prevode i pokreću naredbom `mvn test`. Naredbom `mvn test-compile` može se samo prevesti testove bez pokretanja. `.jar` dokument se generira naredbom `mvn package`, a generirani artefakt se može instalirati u lokalni repozitorij naredbom `mvn install`.

Apache Maven

Osnovna struktura *Maven* projekta:

```
my-app
|-- pom.xml
‘-- src
    |-- main
    |   ‘-- java
    |       ‘-- com
    |           ‘-- mycompany
    |               ‘-- app
    |                   ‘-- App.java
    ‘-- test
        ‘-- java
            ‘-- com
                ‘-- mycompany
                    ‘-- app
                        ‘-- AppTest.java
```

Apache Maven

Promjene u izgradnji projekta se vrše **dodavanjem** ili **rekonfiguriranjem** dodataka:

```
...
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.3</version>
      <configuration>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>
  </plugins>
</build>
<!-- omoguci koristenje JDK 5.0 izvornih datoteka -->
```

Apache Maven

Resursi se dodaju u `.jar` datoteku preko strukture direktorija projekta.

```
my-app
|-- pom.xml
‘-- src
    |-- main
    |   |-- java
    |   |   ‘-- com
    |   |       ‘-- mycompany
    |   |           ‘-- app
    |   |               ‘-- App.java
    |   ‘-- resources
    |       ‘-- META-INF
    |           ‘-- application.properties
    ‘-- test
        ‘-- java
            ‘-- com
```

```
‘-- mycompany
    ‘-- app
        ‘-- AppTest.java
```

Maven omogućava postavljanje **vrijednosti** u neki od dokumenata resursa koja se **dobiva tijekom procesa izgradnje**. Vrijednost može biti neka vrijednost iz `pom.xml` dokumenta, vrijednost definirana u korisničkim postavkama `settings.xml`, svojstvo definirano u eksternom dokumentu koji sadrži svojstva ili među sistemskim svojstvima.

Moguće je definirati ovisnosti o vanjskim resursima koristeći element `<dependencies>`. Odgovarajuće informacije o vanjskom resursu se mogu vidjeti u repozitoriju **Maven Central**. Npr. informacije o paketu `log4j` se mogu vidjeti u direktoriju `/maven2/log4j/log4j` u dokumentu `maven-metadata.xml`.

Apache Maven

Dio `pom.xml` dokumenta koji sadrži definiciju ovisnosti o vanjskim resursima:

```
<dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.12</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
```

Projekt se može spremiti u **udaljeni repozitorij** navođenjem:

```
<distributionManagement>
 <repository>
   <id>moj-repositorij</id>
   <name>Moj Repositorij</name>
   <url>scp://repository.mycompany.com/repository/maven2</url>
 </repository>
</distributionManagement>
```

Maven se može koristiti i za izgradnju **više projekata odjednom**. U tom slučaju treba definirati jedan glavni POM dokument koji navodi projekte kao **module**.

Apache Maven

Primjer strukture direktorija kod projekata s više modula:

```
+- pom.xml
+- my-app
| +- pom.xml
| +- src
|    +- main
|       +- java
+- my-webapp
| +- pom.xml
| +- src
|    +- main
|       +- webapp
```

Dokument `pom.xml` treba sadržavati:

```
<modules>
    <module>my-app</module>
    <module>my-webapp</module>
  </modules>
```

Apache Maven

Ovisnost o aplikaciji `my-app` unutar `my-webapp` možemo definirati unutar dokumenta `my-webapp/pom.xml`.

```
<dependencies>
    <dependency>
      <groupId>com.mycompany.app</groupId>
      <artifactId>my-app</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
    ...
  </dependencies>
```

Također, treba u `pom.xml` dokumente oba pod-projekta dodati:

```
<parent>
    <groupId>com.mycompany.app</groupId>
    <artifactId>app</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
```

Apache Maven

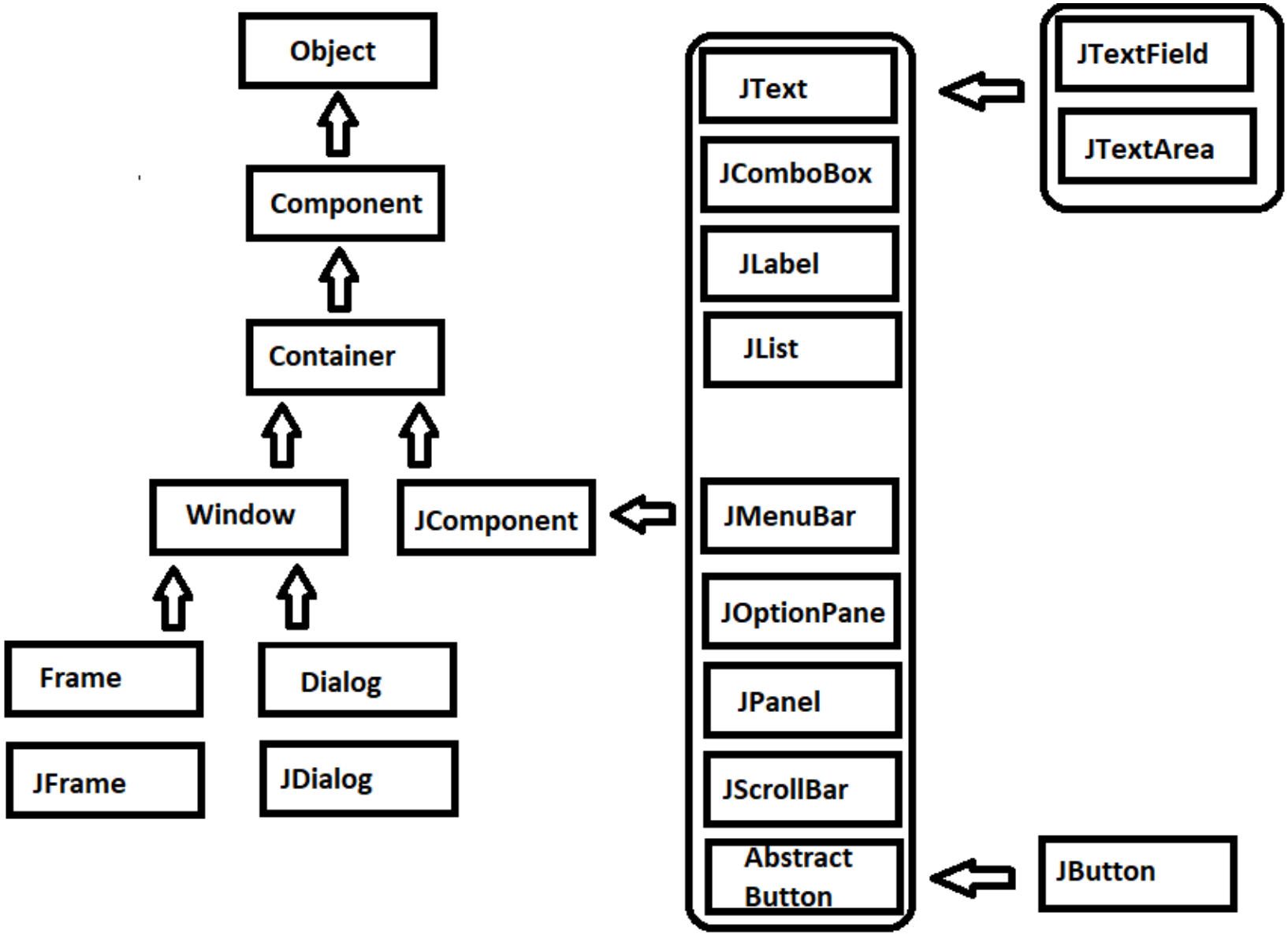
Unutar okruženja *Apache Netbeans* možemo stvoriti aplikaciju koja koristi *Maven* za kreiranje izvršne datoteke (tzv. *Java with Maven* projekt).

Usporedba aplikacijske logike *Java Swing* i *JavaFX*

Java Swing i *JavaFX* su **alati za stvaranje aplikacija s korisničkim sučeljem** u programskom jeziku *Java*. Bazirani su na **prozorima** koji sadrže **skup kontrola** preko kojih se **mogu izvoditi predefinirane akcije** (tu funkcionalnost koristi većina modernih operacijskih sustava). *Java Swing* sadrži glavnu dretvu koja se zove **dretva pogonjena događajima**, ona reagira na razne događaje koji se odvijaju pri **interakciji korisnika s kontrolama korisničkog sučelja**.

Java Swing je razvijen kao sustav koji zamjenjuje i proširuje *AbstractWindowToolkit* (*AWT*). Za razliku od *AWT*, *Java Swing* je neovisan o platformi, fleksibilniji, nudi veći skup komponenti za stvaranje korisničkog sučelja, omogućuje mijenjanje izgleda sučelja tijekom izvršavanja i slijedi paradigmu `model-pogled-kontroler` (*MPK*) (eng. *model-view-controller* ili *MVC*). Model **reprezentira podatke aplikacije**, pogled **vizualnu reprezentaciju podataka** a kontroler **dohvaća korisničke akcije** (naredbe, ulaze) i **prevodi ih u promjene modela** (povezuje model i pogled).

Arhitektura *Java Swing*-a



Arhitektura *Java Swing*-a

U implementaciji *Java Swing*-a su pogled i kontroler spojeni u jedan objekt **korisničkog sučelja**. Ta podjela u kojoj je model odvojen a pogled i kontroler su spojeni u jedan objekt se naziva **arhitektura odvojivog modela**. *Java Swing* prati dobru praksu da se **arhitektura aplikacije centrira oko podataka** na način da **pruža posebno sučelje modela** za svaku komponetu koja **ima logičke podatke ili apstrakcije vrijednosti**. Ta podjela omogućuje da se unutar programa definiraju posebne implementacije modela ovisne o komponentama i njihovoj funkcionalnosti. *Swing* modeli pripadaju jednoj od dvije generalne kategorije: **modeli stanja grafičkog sučelja** i **modeli podataka aplikacije**.
Modeli stanja grafičkog sučelja su sučelja koja **definiraju vizualna stanja kontrole grafičkog sučelja** (je li gumb pritisnut, koji elementu su selektirani na listi itd.). Modeli stanja grafičkog sučelja su **relevantni samo u kontekstu grafičkog korisničkog sučelja**. Dosta često je **poželjno koristiti ovu vrstu modela**, pogotovo ukoliko je više kontrola povezano u zajednička stanja ili ako **manipuliranje jednom kontrolom mijenja neku drugu kontrolu**.

Arhitektura *Java Swing*-a

Unatoč tome, moguće je manipulirati stanjem kontrole grafičkog sučelja i **kroz metode visokog nivoa nad komponentom** (bez interakcije s modelom).

Modeli podataka aplikacije su sučelja koja **reprezentiraju neke podatke** i imaju značenje u kontekstu aplikacije (vrijednost elementa tablice, elementi u listi). Takvi modeli pružaju snažnu programsku paradigmu za *Swing* programe koji trebaju **jasnu separaciju između podataka aplikacije, logike i grafičkog sučelja**. Za *Swing* komponente jako povezane s podacima kao što su `JTree` ili `JTable` se snažno preporuča interakcija s podatkovnim modelom. Komponente kao `BoundedRangeModel`, `JSlider` ili `JProgressBar` se nalaze između modela podataka aplikacije i modela stanja grafičkog sučelja ovisno o kontekstu u kojem se model koristi. *Swing* odvojivi model **ne radi razliku** između modela stanja grafičkog sučelja i modela podataka aplikacije.

Baza *Java Swing* aplikacije je **prozor** (`JFrame`) koji **sadrži naslov i kontrole za minimizaciju, maksimizaciju i gašenje prozora**.

Arhitektura *Java Swing*-a

`JDialog` je klasa koja reprezentira **skočni prozor**, a `JPanel` je klasa koja **reprezentira kontejner koji služi za organizaciju komponenti na prozoru** (prozor može sadržavati više panela).

Ukoliko koristimo model **istog tipa** unutar više komponenti, **njihova stanja će biti automatski povezana**. Ukoliko programer ne postavi model komponenti, **bit će joj pridijeljen standardni model**.

Modeli **moraju moći obavijestiti odgovarajuće instance kada je došlo do promjene njihovih podataka ili vrijednosti**. *Swing* modeli za implementaciju javljanja te informacije koriste **model *JavaBeans* događaja**. Javljanje promjena se može napraviti na dva načina: a) **slanjem jednostavne i kratke poruke** da je došlo do promjene stanja i da se zahtijeva od zainteresiranog objekta koji osluškuje takve poruke da odgovori porukom da bi saznao do kakvih promjena je došlo. Prednost takvog pristupa je što se **jedna instanca događaja** može koristiti za **slanje svih obavijesti nekog modela** (poželjno u slučajevima kada dolazi do slanja velikog broja obavijesti).

Arhitektura *Java Swing*-a

Drugi način je **slanje poruke koja sadrži stanje i opisuje preciznije kako se model promijenio**. Taj način **zahtijeva stvarnje nove instance događaja za svaku obavijest**. Korištenje ovog načina je **poželjno kada generičke obavijesti ne pružaju dovoljno informacija objektu koji osluškuje da bi mogao učinkovito odrediti što se promijenilo u modelu** (npr. kada stupac elemenata promijeni vrijednost u *JTable*).

Model ima **listu objekata koji osluškuju promjenu njegovog stanja**. *Swing* komponenta se **brine o spajanju odgovarajućeg objekta koji osluškuje** tako da se **odgovarajuća komponenta može iscrtati ako se model promijeni**.

Svaka komponenta koja se koristi u aplikaciji ima **standardno postavljen izgled i funkcionalnost reagiranja na događaje** (obavijesti) - ovo se još zove i **osjećaj**. *Swing* dozvoljava da se ta funkcionalnost **promijeni proširivanjem postojećeg izgleda i osjećaja ili stvaranjem novih**.

Arhitektura *Java Swing*-a

Kreiranje novog izgleda i osjećaja znači da se **odgovarajući dio implementacije delegira posebnom objektu korisničkog sučelja koji je trenutno postavljen i može se promijeniti tijekom izvođenja**. API za izmjenu izgleda i osjećaja sadrži: a) određene metode (tzv. kuke - koje mogu presresti funkcijske pozive, poruke ili događaje koji se izmjenjuju između komponenata), b) API visokog nivoa za upravljanje izgledom i osjećajem, c) kompliciraniji API koji implementira izgled i osjećaj u posebnim paketima.

Swing dopušta aplikacijama **definiranje vrijednosti svojstava komponente** (npr. boju, font). Ukoliko se neispravno koristi (nakon konstrukcije komponente) u kombinaciji s promjenom izgleda i osjećaja, može doći do poništavanja svojstava od strane promjene izgleda i osjećaja. Zbog toga *Swing* označava sve vrijednosti postavljene od strane promjene izgleda i osjećaja sučeljem `plaf.UIResource`. Klasa `swing.plaf.ComponentUI` **sadrži mehanizme koji omogućavaju promjenu izgleda i osjećaja**.

Arhitektura *Java Swing*-a

Metode te klase omogućuju instalaciju i deinstalaciju korisničkog sučelja i delegaciju obrade geometrije i crtanja komponente. **Delegat** `instalUI()` korisničkog sučelja je odgovoran za: a) postavljanje standardnog fonta, boje, granica i neprozirnosti komponente, b) instaliranje odgovarajućeg upravljača razmještaja komponenata, c) dodavanje podkomponenata registru komponente i odgovarajućih osluškivača događaja, d) registracija akcija tipkovnice (mnemonika i slično) povezanih s izgledom i osjećajem, e) registracija odgovarajućih osluškivača modela koji dojavljaju kada treba ponovo iscrtati komponentu, f) inicijalizacija podataka.

`LayoutManager` je **zadužen za raspoređivanje komponenata po zadanom algoritmu**. Obično se za to koristi informacija o preferiranoj veličini (ponekad minimalnoj i/ili maksimalnoj ovisno o algoritmu) da bi se odredilo kako točno **pozicionirati te komponente i koju konačnu veličinu im postaviti**.

Arhitektura *Java Swing*-a

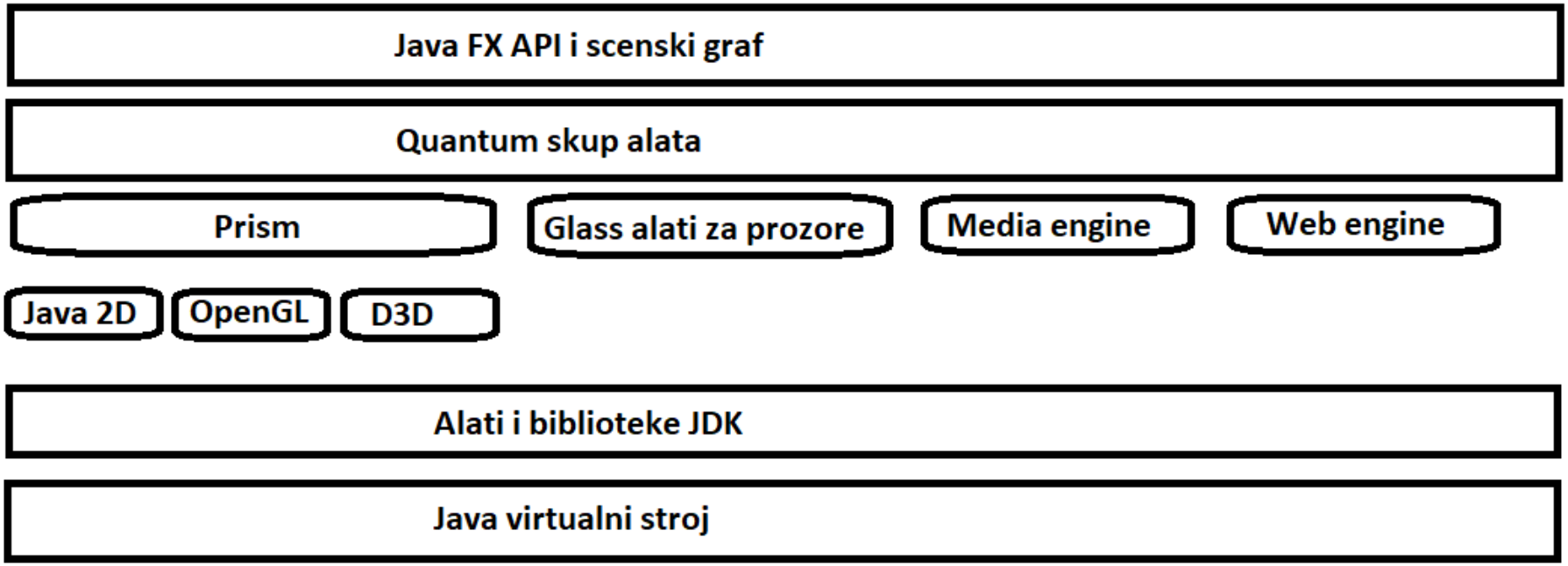
Sve metode na `ComponentUI` primaju `JComponent` objekt kao parametar. Zbog toga se delegat korisničkog sučelja može implementirati i **bez informacije o stanju**. Razlog je što delegat **može uvijek ispitati stanje neke komponente**. Takav način implementacije delegata **omogućava stvaranje samo jedne instance delegata korisničkog sučelja za sve instance klase te komponente** što može znatno reducirati broj kreiranih objekata. Takav pristup **funkcionira dobro za manje komponente grafičkog sučelja**, međutim **nije prikladan za kompleksnije komponente** za koje se ipak predlaže korištenje delegata koji sadrže informaciju o stanju objekta. Razlog je što je **trošak kreiranja više objekata delegata manji od konstantnog računanja stanja objekta**. Ovdje se uzima u obzir i činjenica da je **broj kompleksnih objekata u programu obično malen**.

Usporedba aplikacijske logike *Java Swing* i *JavaFX*

JavaFX je noviji programski alat za stvaranje korisničkih sučelja (potencijalni nasljednik *Swinga*) koji je baziran na konceptu **scenskoga grafa**. Element scenskoga grafa se zove **čvor** i ima *ID*, klasu stila i obuhvačajući volumen (volumen prostora koji obuhvaća ovaj objekt i sve povezane objekte). Svaki čvor u scenskom grafu osim korijena ima **jednog roditelja i nula ili više djece**. Također može imati: a) efekte (npr. zamučivanje i sjenčanje), b) prozirnost, c) transformacije, d) obrađivače događaja, e) stanje ovisno o aplikaciji. Čvorovi mogu biti različiti tipovi objekata kao što su $2D$ ili $3D$ objekti, slike, medija, ugrađeni preglednik, tekst, kontrole korisničkog sučelja, grafovi, grupe i spremnici.

Za razliku od *Swinga* i *AWT*-a, scenski graf *JavaFX*-a uključuje i osnovne grafičke objekte kao što su četverokuti i tekst. Sadrži i kontrole, spremnike slojeva, slike i multimediju. Koncept scenskog grafa pojednostavljuje rad s korisničkim sučeljem i omogućava **visoku razinu dinamičnosti sadržaja**, stoga je moguće unutar korisničkog sučelja imati **animacije** (za što *Swing* nije dizajniran iako je moguće napraviti određene jednostavnije animacije).

Arhitektura *JavaFX*-a



Platforma *JavaFX* **kombinira** mogućnosti *Jave* i vizualne funkcionalnosti *JavaFX*-a kroz programsko sučelje. To programsko sučelje **omogućuje korištenje generičkih metoda i klasa**, **anotacija**, **višedretvenih konstrukcija**, **lambda izraza**. Olakšano je korištenje *JavaScript*-a (moguće i korištenje *CSS*-a). Omogućuje korištenje **uočljivih lista i mapa** koje **dozvoljavaju povezivanje korisničkog sučelja i podatkovnog modela, uočavanje promjena u podatkovnom modelu i ažuriranje odgovarajućih komponenti korisničkog sučelja**.

Arhitektura *JavaFX*-a

Skup alata *Quantum* **integrira skup alata za rad s prozorima** *Glass* (generira i pulseve za animaciju, površine, sadrži visokorezolucijske štoperice ovisne o platformi, održava red događaja koristeći sistemski red događaja, ovisan o platformi), **grafičku komponentu** *Prism* koja sadrži kod za iscrtavanje prozora, te pruža njihove funkcionalnosti sloju *JavaFX*. Grafički sustav *JavaFX*-a nudi mogućnost softverskog iscrtavanja ukoliko nije dostupan odgovarajući grafički hardver na računalu. U istom sloju se nalaze i *media engine* **komponenta zadužena za integraciju audia i videa** te *web engine* **zadužena za integraciju web funkcionalnosti** u *JavaFX* aplikaciju. Grafička komponenta koristi alate *Java* 2*D* (alat za 2*D* grafiku), *OpenGL* (grafički standard koji omogućava kreiranje raznih grafičkih elemenata na procesoru i/ili grafičkoj kartici računala, podržava 2*D* i 3*D* grafiku) i *D3D* (*Direct 3D*, grafičko sučelje za grafiku na operacijskom sustavu *Windows*, dio platforme *DirectX*).

Arhitektura *JavaFX*-a

Pri radu *JavaFX* aplikacija izvode se **dvije ili više dretava različitog tipa**. Glavna aplikacijska dretva **pristupa svim aktivnim scenama**. Graf scene se može kreirati i modificirati u **pozadinskoj dretvi**, ali kada se korjenski čvor tog grafa pridruži nekom aktivnom objektu na sceni, tada tom grafu pristupa aplikacijska *JavaFX* dretva. To omogućava izradu kompleksnih grafova scena bez komprimitiranja rada animacije i sučelja. Glavna dretva *JavaFX*-a je drugačija od glavne dretve *Swing*-a stoga treba pažljivo pristupiti integraciji tih pristupa.

Prism dretva crtač radi nezavisno od dodijeljivača događaja. Dozvoljava iscrtavanje nekog okvira dok se drugi računa. Podržava korištenje višestrukih rasterizacijskih dretvi koje pomažu pri složenim računima.

Dretva medije se izvršava u pozadini i sinkronizira najnovije okvire kroz scenski graf koristeći glavnu *JavaFX* aplikacijsku dretvu.

Arhitektura *JavaFX*-a

Pulse je događaj koji dojavljuje *JavaFX* scenskom grafu da je vrijeme za sinkronizaciju stanja elemenata od strane sustava *Prism*. Pulse se odvija maksimalnom brzinom od 60 okvira po sekundi i aktivira se svaki puta kada se izvodi animacija na scenskom grafu. Puls se aktivira i kada dođe do promjene na sceni (npr. promjena pozicije gumba). Nakon pokretanja pulsa se sinkronizira stanje elemenata grafa scene skroz do sloja koji radi iscrtavanje. Puls omogućava asinkronu obradu događaja, što omogućuje skupljanje događaja i njihovo izvršavanje na pulsu. Poredak i struktura elemenata sučelja i *CSS* su također povezani s pulsom (dolazi do njihovih promjena pri promjenama čvorova grafa).

Web komponenta je bazirana na *Webkit*-u koji kroz svoj API omogućuje korištenje Internet preglednika s punom funkcionalnosti (*HTML*5, *CSS*, *JavaScript*, *DOM*, *SVG*). Ugrađeni preglednik se sastoji od klase `WebEngine` (nudi osnovnu funkcionalnost pregledavanja stranica) i `WebView` (enkapsulira `WebEngine` objekt, ubacuje *HTML* sadržaj unutar aplikacijske scene i pruža elemente članove i metode za primjenu efekata i transformacija).

Arhitektura *JavaFX*-a

`WebView` nasljeđuje klasu `Node`. *JavaFX* omogućava kontrolu *Java* poziva iz *JavaScript*-a (i obrnuto) čime se **omogućava korištenje funkcionalnosti oba jezika**.

CSS **omogućava izmjenu stila korisničkog sučelja** *JavaFX* aplikacije bez izmjene izvornog koda aplikacije. Može se **asinkrono primijeniti na proizvoljan čvor scenskog grafa**. Moguće je primijeniti *CSS* dinamički (tijekom izvođenja) što **omogućava dinamičku promjenu izgleda aplikacije**.

Kontrole *JavaFX*-a su **portabilne** i **grade se koristeći čvorove scenskog grafa**. Kontrole kao *accordion*, *check box*, *color button*, *list view*, *progress bar itd.* se nalaze u paketu `javafx.scene.control`.

Raspored kontrola na sučelju (eng. *layout*) se može koristiti za **fleksibilan i dinamičan raspored kontrola korisničkog sučelja unutar scenskog grafa aplikacije**.

Arhitektura *JavaFX*-a

API vezan uz raspored kontrola sadrži kontejnere `BorderPane` (klasa rasporedi sadržaj u gornjem, donjem, desnom, lijevom ili centralnom dijelu), `HBox` (klasa raspoređuje sadržaj horizontalno u jednom redu), `VBox` (klasa raspoređuje sadržaj vertikalno u jednom stupcu), `StackPane` (klasa stavlja čvorove sadržaja u stog, klikom na kontrolu se mijenja čvor sa sadržajem), `GridPane` (omogućava korištenje fleksibilne mreže redaka i stupaca koje se mogu smjestiti čvorovi), `FlowPane` (uređuje čvorove horizontalno ili vertikalno u niz koji je ograničen duljinom ili širinom, prelazi u novi redak/stupac kada se dosegne limit), `TilePane` (uređuje sadržaj u jednako raspoređena polja ili regije), `AnchorPane` (omogućava da se čvor usidri s gornje, donje, lijeve strane ili iz sredine). **Moguće je ungijezditi više različitih rasporeda kontrola** da bi se dobio željeni konačni raspored.

Svaki čvor u scenskom grafu se **može transformirati** u koordinatnom sustavu koristeći `translate` (translatira čvor po x, y ili z osi), `scale` (skalira čvor za određeni faktor), `shear` (rotira jednu os tako da x-os i y-os više nisu okomite, mijenja koordinate čvora koristeći specificirane multiplikatore),

Arhitektura *JavaFX*-a

`rotate` (rotira čvor oko zadane točke na sceni koja se zove *pivot*), `affine` (vrši linearno mapiranje $2D/3D$ koordinata u druge $2D/3D$ koordinate čuvajući svojstva ravnih i paralelnih linija). Ove metode su definirane u paketu `javafx.scene.transform`.

Efekti u *JavaFX*-u su **primarno bazirani na pikselima slika** stoga obuhvate skup čvorova iz scenskog grafa, renderiraju ih kao sliku i primjene na nju zadane efekte.

Neke klase koje pružaju vizualne efekte unutar *JavaFX*-a su: `Drop Shadow` (iscrta sjenu sadržaja iza sadržaja na koji se primijeni), `Reflection` (iscrtava reflektiranu verziju sadržaja ispod tog sadržaja), `Lighting` (simulira izvor svijetlosti koji sja na zadani sadržaj, može ravnom objektu pružati realističniji, trodimenzionalni oblik).