Nezavisne biblioteke

Autori:
Goran Savić
Milan Segedinac

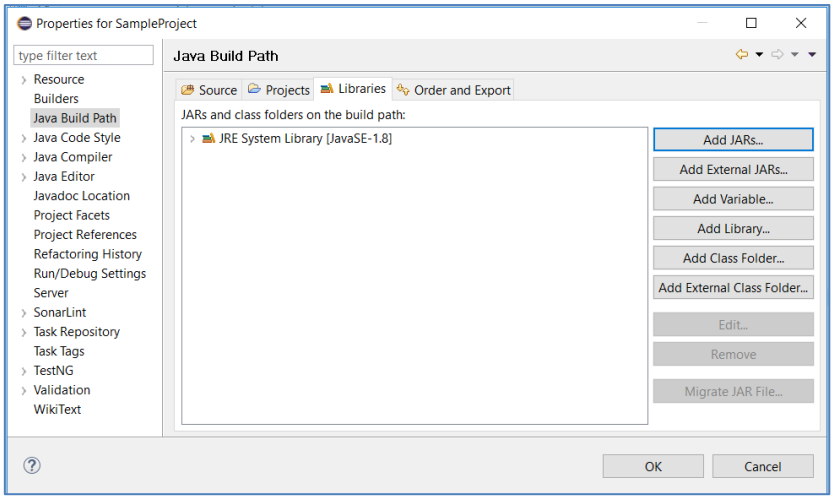
1. Nezavisne biblioteke

Do sada smo videli da značajan deo funkcionalnosti programa možemo realizovati korišćenjem postojećih klasa koje su deo Java biblioteke klasa. Pored ovih klasa, postoji i veliki broj potpuno nezavisno razvijenih i distribuiranih Java klasa koje se mogu koristiti u različitu svrhu. Obzirom da se skup klasa koje zajedno obezbeđuju određenu funkcionalnost naziva biblioteka klasa, ovakve nezavisno razvijane skupove klasa nazivamo nezavisna biblioteka (eng. *third party library*).

Biblioteke klasa se u Javi distribuiraju u formi fajlova sa ekstenzijom JAR. To su ustvari ZIP arhive, samo imaju drugačiju ekstenziju da bi bilo očiglednije da je reč o Java bibliotekama. Na ovaj način Java klase se distribuiraju kompresovane. Takođe, pored Java klasa, biblioteka može da sadrži dodatne fajlove za procesiranje biblioteke. ZIP format je široko prepoznat, pa je time obezbeđena podrška za rad sa Java bibliotekama. Jedna biblioteka može da grupiše klase u više jar fajlova. Takođe, klase koje se nalaze u jar fajlu mogu da u svom radu koriste klase iz drugih biblioteka. Tada kažemo da jar fajl zavisi od drugih jar fajlova. Za ispravan rad biblioteke, neophodno je obezbediti dostupnost jar fajlova u kojima su klase sa funkcionalnostima biblioteke, kao i svih jar fajlova od kojih biblioteka zavisi.

Što se tiče dostupnosti nezavisnih klasa iz Java programa, za ove klase važi isto što i za bilo koje druge klase. Kao što smo ranije objasnili, da bi klasa bila dostupna u programu, neophodno je da se nalazi na nekoj od lokacija na kojima JVM traži klase koje se referenciraju u programu. Ove lokacije su definisane u CLASSPATH parametru programa (ili CLASSPATH promenljivoj okruženja ako ne postoji parametar definisan specifično za projekat). Da bi klase iz nekog JAR fajla bile dostupne, neophodno je i da taj JAR fajl bude dodat u spisak lokacija u CLASSPATH promenljivoj.

Spominjali smo da Eclipse IDE ima podršku za postavljanje CLASSPATH promenljive za projekat. U konfiguraciji projekta, može se dodati određeni JAR fajl sa diska u CLASSPATH. Ovo se vrši dodavanjem fajla u *build path* projekta u podešavanjima projekta. Slika prikazuje podešavanja *build path* za projekat u Eclipse IDE. JAR fajl se u *build path* dodaje klikom na dugme *Add JARs*, ukoliko se fajl nalazi unutar projekta. Druga varijanta je klik na dugme *Add External JARs* za JAR fajlove koji su na nekoj drugoj lokaciji na disku.



Nezavisne biblioteke

Realni projekti koriste veliki broj nezavisnih biblioteka, pa se obično za obezbeđivanje dostupnosti ovih biblioteka koriste specijalizovani alati. Ovaj deo podrške razvoju projekta se naziva upravljanje zavisnostima (eng. *dependency management*). Alati koji se najčešće koriste za upravljanje zavisnostima u Java programima su Apache Maven i Gradle.

Obzirom da je zajednica Java programera veoma velika, postoji mnogo nezavisnih biblioteka koje omogućuju realizaciju najrazličitijih funkcionalnosti programa. Kao primer jedne nezavisne biblioteke ovde ćemo ukratko predstaviti biblioteku pod nazivom Apache POI. Ova biblioteka sadrži klase koje omogućuju programsku obradu Microsoft Office dokumenata iz Java programa. Od podrške za različite tipove Office dokumenata, ovde ćemo predstaviti samo deo biblioteke koji omogućuje rad sa Microsoft Excel dokumentima. POI podržava rad sa .xls i .xlsx fajlovima kroz svoje potprojekte HSSF, odnosno XSSF. Ovde ćemo analizirati XSSF obzirom da podžava noviji format Excel dokumenata.

Podrazumeva se da nije moguće poznavati detalje rada nezavisnih biblioteka bez konsultovanja dokumentacije. Većina nezavisnih biblioteka ima dokumentaciju koja opisuje njen API, a najčešće su dostupni i primeri korišćenja biblioteke. Pomenuta dva izvora informacija su osnov za rad sa svakom nezavisnom bibliotekom. Primena biblioteke za željeni scenario se vrši analizom API-ja, primera i ostale dokumentacije.

Pomenuto važi i za POI biblioteku. Na internet adresi https://poi.apache.org se nalazi zvaničan sajt ove biblioteke. Na ovom sajtu se nalazi dokumentacija za korišćenje biblioteke. Takođe, na sajtu je moguće preuzeti jar fajlove koji čine biblioteku. Pogledajmo najznačajnije klase i interfejse iz XSSF POI projekta.

`Workbook`	Interfejs koji predstavlja Excel dokument
`XSSFWorkbook`	Implementacija `Workbook` interfejsa koja predstavlja .xlsx dokument
`Sheet`	Interfejs koji predstavlja list Excel dokumenta
`XSSFSheet`	Implementacija `Sheet` interfejsa koja predstavlja list .xlsx dokumenta
`Row`	Interfejs koji predstavlja red u Excel tabeli
`XSSFRow`	Implementacija `Row` interfejsa za .xlsx dokument
`Cell`	Interfejs koji predstavlja ćeliju Excel tabele
`XSSFCell`	Implementacija `Cell` interfejsa za .xlsx dokument

U nastavku ćemo prikazati na koji način se prikazane klase i interfejsi koriste za rad sa Excel dokumentima. Prikazaćemo jednostavan primer programskog kreiranja Excel dokumenta i popunjavanja dokumenta podacima o državama.

Prvi korak u realizaciji pomenutog zadatka je programsko kreiranje dokumenta i lista u dokumentu. Dat je kod koji realizuje ovu funkcionalnost.

```
Workbook wb = new XSSFWorkbook();
Sheet sheet = wb.createSheet("Drzave");
```

Prikazani kod kreira objekat koji predstavlja novi Excel dokument. Dokument sadrži jedan list pod nazivom Države. U nastavku je prikazan kod koji kreira redove na ovom listu i popunjava ćelije ovih redova podacima o državama.

```
Row row = sheet.createRow(0);
Cell cell = row.createCell(0);
cell.setCellValue("Oznaka");
cell = row.createCell(1);
cell.setCellValue("Naziv");

row = sheet.createRow(1);
cell = row.createCell(0);
cell.setCellValue("sr");
cell = row.createCell(1);
cell.setCellValue("Srbija");

row = sheet.createRow(2);
cell = row.createCell(0);
cell.setCellValue("fr");
cell = row.createCell(1);
cell.setCellValue("Francuska");
```

U kodu su ukupno kreirana tri reda, pri čemu prvi red predstavlja zaglavlje tabele. Ćelije naredna dva reda sadrže podatke o državama, konkretno o Srbiji i Francuskoj.

Definisani dokument za sada postoji samo u radnoj memoriji. Naredni kod pokazuje kako se može kreirani Excel dokument snimiti na disk kao xlsx fajl. Vidimo da se za ovu svrhu koristi metoda `write` definisana u klasi `XSSFWorkbook`.

```
try {
	FileOutputStream fileOut = new FileOutputStream("drzave.xlsx");
	wb.write(fileOut);
	fileOut.close();
} catch (FileNotFoundException e) {
	e.printStackTrace();
} catch (IOException e) {
	e.printStackTrace();
}
```