1. domaća zadaća

Prije no što krenete dalje, pročitajte uputu na kraju dokumenta.

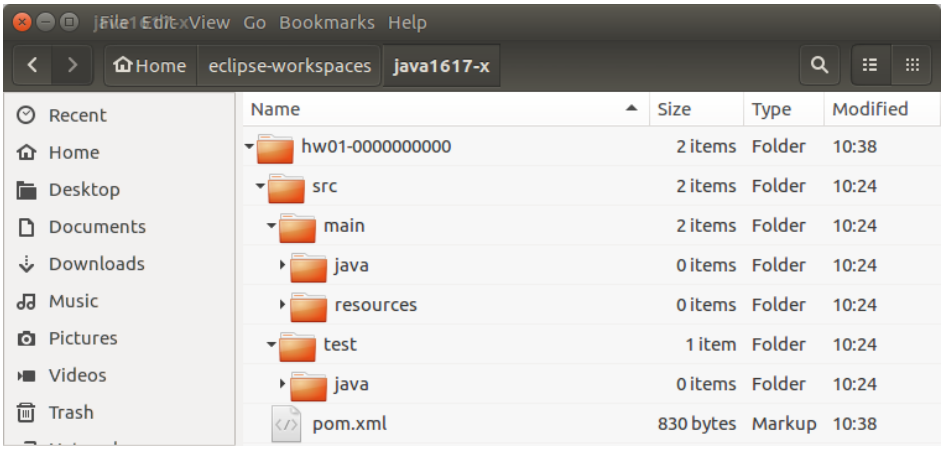
1. Početno podešavanje

Pokrenite *Eclipse* i odaberite *workspace* direktorij (ako ćete koristiti neki drugi a ne onaj koji ste već napravili na predavanjima). Kad se *Eclipse* pokrene, na disku će napraviti taj direktorij, i u njemu (vjerojatno) poddirektorij `RemoteSystemsTempFiles` kao i skriveni poddirektorij `.metadata`.

Sljedeće radite izvan *Eclipsa* (izravno kroz alate operacijskog sustava).

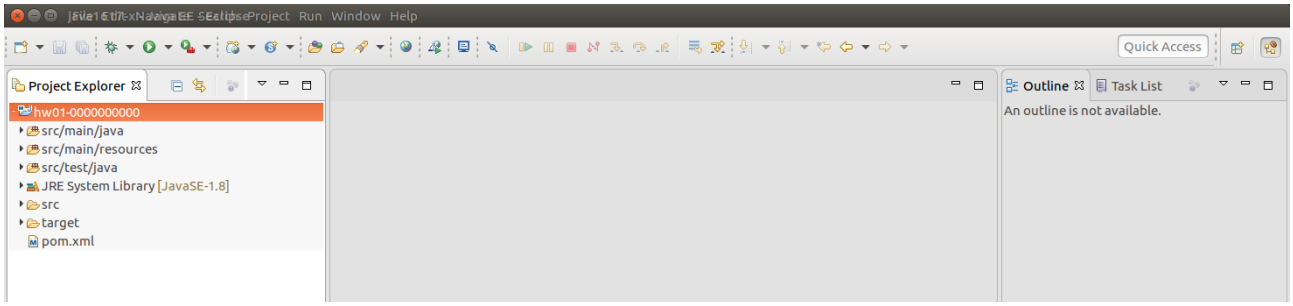
1. U *Eclipse* workspace-u na disku napravite novi direktorij:
   `hw01-0000000000` (zamijenite nule Vašim JMBAGom)
2. U njega stavite osnovnu verziju datoteke `pom.xml` (te deklarirajte ovisnosti prema biblioteci `junit` kako smo to napravili na predavanju). Istu otvorite u uređivaču teksta i osigurajte da je `groupID` postavljen kao:
   `hr.fer.zemris.java.jmbag0000000000` (zamijenite nule Vašim JMBAGom)
   a `artifactID` da je:
   `hw01-0000000000` (zamijenite nule Vašim JMBAGom).
3. Potom napravite u direktoriju projekta još poddirektorije:
   `src/main/java`
   `src/main/resources`
   `src/test/java`

Stanje na disku trebalo bi odgovarati prikazanome na sljedećoj slici.



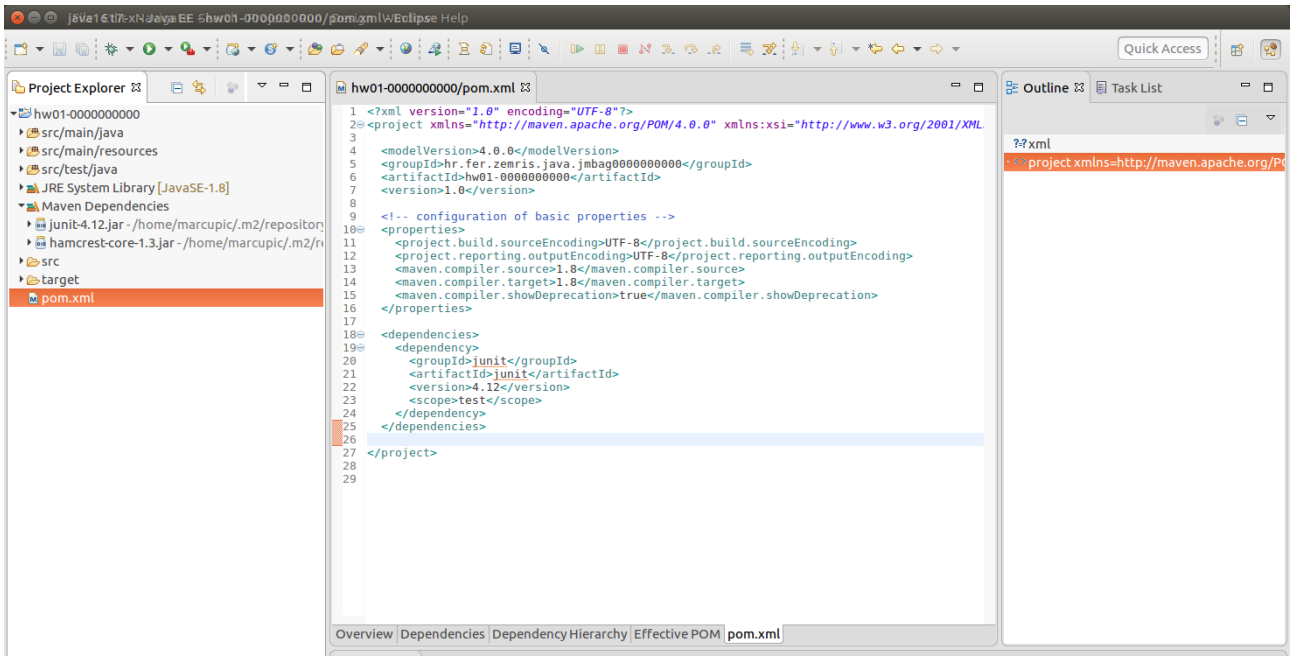
Sada se ponovno vratite u *Eclipse*. Odaberite iz izbornika `File` → `Import ...` → `Maven` → `Existing Maven Projects`, `Next`, kao `Root directory` (desno klik na gumb `Browse...`) odaberite direktorij projekta za domaću zadaću (`hw01-xxxxxxxxxx`). U popisu projekata koji će se potom prikazati označite (ako već nije) `/pom.xml` koji odgovara ovoj domaćoj zadaći (bit će Vam prikazan `groupID:artifactId` - provjerite je li u oba korektno evidentiran Vaš JMBAG; ako nije, `Cancel` pa izvana u editoru korigirajte `pom.xml` i potom ponovite postupak `Import`). Ako je sve u redu, klik na gumb `Finish`.

Trebali biste dobiti prikaz kao na sljedećoj slici (uz razliku što će pisati JavaSE-11 ili nešto na tu temu, te što nije prikazana sekcija *Maven Dependencies* koja je prikazana na slici iza).



Maven projekt time će se prikazati u *Project Exploreru* (lijevi dio prozora u *Java EE* perspektivi). Ekspandirajte projekt pa dvoklik na `pom.xml` i kad se otvori onaj grafički prikaz, pri dnu ćete imati karticu "`pom.xml`" - klik na nju i prikazat će se tekst datoteke `pom.xml`. Uvjerite se da je unutra prisutna sekcija `<dependencies>`, i unutar toga ovisnosti prema biblioteci `junit` koje imaju `scope` postavljen na `test`. Ako to nije slučaj, popravite pa pohranite izmjene (CTRL+S).

U stavkama projekta postoji i sekcija *Maven Dependencies*, te će unutar toga biti nekoliko biblioteka (slika u nastavku prikazuje stanje od godine kada je bio korišten junit verzije 4.12; ove godine koristimo junit verzije 5.4 pa je iza sljedeće slike dodan screenshot koji odgovara tome). Varijable koje su prikazane u sekciji `<properties>` na donjoj slici a nisu kod Vas zanemarite, kao i pogrešne prikazane verzije (slika je od ranijih godina).



Uz junit verzije 5.4:

Sada ste spremni za rješavanje zadataka koji slijede.

Kada radite prvi program, napravite mišem desni klik u *Project Exploreru* na naziv direktorija u koji treba smjestiti izvorni kod (`*.java` datoteke) – u našem slučaju na `src/main/java`, pa u iskočnom izborniku odaberite `New → Class`. U prozoru koji se otvori, pri vrhu imate mjesto gdje možete upisati naziv paketa u koji treba smjestiti razred/program a malo ispod toga i mjesto gdje možete upisati naziv razreda/programa.

Prilikom pisanja koda, pridržavajte se sljedećih konvencija.

- Nazivi razreda/sučelja/enumova uvijek se pišu velikim početnim slovom. Ako su konkatenacija više riječi, svaka sljedeća opet započinje velikim početnim slovom. Npr. `Student`, `StudentMail`, `StudentCourseEnrolment`.
- Nazivi metoda uvijek se pišu malim početnim slovom. Ako su konkatenacija više riječi, svaka sljedeća opet započinje velikim početnim slovom. Npr. `print`, `printGrades`, `printStudentGrades`. Nikada nemojte kratiti imena (primjerice izbacivanjem suglasnika, što zna biti praksa u C kodu; dakle ne: `prnt`, `prntGrds` i slično). Također, riječi nikad ne spajajte podvlakama (također praksa iz C-a); znači ne: `print_student_grades`.
- Nazivi varijabli slijede istu konvenciju kao i metode: `dayOfMonth`, `monthOfYear`, itd.
- Varijable deklarirajte uvijek tamo gdje ih prvi puta trebate. Nemojte nikada deklarirati sve što Vam treba na početku metode. Varijable najčešće inicijalizirate odmah pri deklaraciji, Dakle, ne:
  ```
  int month;
  month = 7;
  ```
  već
  ```
  int month = 7;
  ```
  Također, ne:
  ```
  int month;
  month = calculateMonth();
  ```
  već:
  ```
  int month = calculateMonth();
  ```
- Doseg varijable treba biti što je manji (uži) moguć. Ovo je korektno:
  ```
  int i;
  for(i = 0; i < 10; i++) {
   ...
  }
  ```
  ali ako nam taj `i` ne treba iza petlje `for`, onda je puno bolje:
  ```
  for(int i = 0; i < 10; i++) {
   ...
  }
  ```
- Nove varijable mogu se definirati i unutar blokova. Primjerice, ako nam treba varijabla koja postoji samo u tijelu naredbe `if`, tamo je deklarirajte. Npr.
  ```
  if(month > 6) {
    double bonusFactor = 1.2;
    ... radi nešto s bonusFactor ...
  }
  ```
- Obavezno korektno indentirajte kod!
- Obavezno pišite Javadoc kako smo objasnili na predavanju.
- Uočite kako smo u prethodnim primjerima radili s vitičastim zagradama. Vitičasta zagrada otvara se u nastavku (u istom retku, nakon jedne praznine) naredbe kojoj taj blok pripada (pogledajte gore primjer naredbi `if` i `for`), u sljedećim retcima uvučeno dolaze naredbe koje pripadaju tom bloku, i potom u novom retku vizualno poravnatno okomito prema gore s prvim slovom naredbe kojoj blok pripada dolazi zatvorena vitičasta zagrada (kod `if`-a je poravnata sa slovom `i`, kod `for`-a sa slovom `f`).
- Naredbe `if`, `for`, `do` i while uvijek pišite s pripadnim blokovima, čak ako je unutra i samo jedna naredba. Dakle, ne:

```
if(a > b) printSomething();
```
već:
```
if(a > b) {
  printSomething();
}
```

- Testove uvijek smještajte u `src/test/java`. Naziv uobičajeno odgovara nazivu razreda koji se testira sa sufiksom `Test`; npr. za razred `P07` testovi su u `P07Test`. Metode koje test poziva iz testiranog razreda ne smiju biti `private`.

Zadatak 1.

U paketu `hr.fer.zemris.java.hw01` napravite program `Factorial`. Program se pokreće bez argumenata. Korisnik preko tipkovnice unosi cijele brojeve u rasponu od 3 do 20. Ako korisnik zada broj koji je izvan tog raspona ili ako nije broj, ispisati odgovarajuću poruku i nastaviti dalje s radom. Program ispisuje faktorijelu zadanog broja. Odaberite za izračun primitivni tip podataka uz koji ćete moći izračunati rezultat za ovaj raspon brojeva. Evo očekivanog primjera interakcije programa i korisnika, nakon što se program pokrene:

```
Unesite broj > štefica
'štefica' nije cijeli broj.
Unesite broj > 3
3! = 6
Unesite broj > 3.14
'3.14' nije cijeli broj.
Unesite broj > -4
'-4' nije broj u dozvoljenom rasponu.
Unesite broj > 4
4! = 24
Unesite broj > kraj
Doviđenja.
```

Rad programa prekida se kada korisnik unese "`kraj`". Korisnikovi unosi prikazani su crvenom bojom. Vaš bi program za identične ulaze iz primjera trebao generirati upravo prikazani ispis.

Izračun faktorijele ostvarite u zasebnoj pomoćnoj metodi (složenost nije bitna). Napišite testove za tu metodu. Ako metoda primi vrijednost argumenta za koju ne može izračunati faktorijelu, treba baciti iznimku `IllegalArgumentException`. Provjerite to testovima. Primijetite da se ograničenja na korisniku dozvoljen raspon brojeva općenito (pa tako i ovdje) razlikuju od ograničenja na raspon brojeva koji Vaša metoda zna izračunati. Razmislite gdje ćete u kodu kontrolirati što, i kako ćete organizirati program.

Zadatak 2.

U paketu `hr.fer.zemris.java.hw01` napravite program `Rectangle`. Program pita korisnika preko naredbenog retka da unese širinu pa visinu pravokutnika. Program korisniku ispisuje površinu i opseg pravokutnika. Ako program pri pokretanju dobije argumente preko naredbenog retka, onda ništa ne pita korisnika, već odmah računa i ispisuje površinu i opseg. Korisnik kao širinu i visinu može unijeti i decimalne brojeve. Primjerice, ako program pokrenemo ovako:

```
Rectangle 2 8
```

program će odmah ispisati sljedeće, i nakon toga prekinuti s radom.

```
Pravokutnik širine 2.0 i visine 8.0 ima površinu 16.0 te opseg 20.0.
```

Ako ga pokrenemo bez argumenata, očekivano je ponašanje ilustrirano ispisom u nastavku.

```
Unesite širinu > štefica
'štefica' se ne može protumačiti kao broj.
Unesite širinu > -2.1
Unijeli ste negativnu vrijednost.
Unesite širinu > 2
Unesite visinu > štefica
'štefica' se ne može protumačiti kao broj.
Unesite visinu > -2.1
Unijeli ste negativnu vrijednost.
Unesite visinu > 8
Pravokutnik širine 2.0 i visine 8.0 ima površinu 16.0 te opseg 20.0.
```

Prilikom pisanja programa nemojte sav kod nagurati u metodu `main`. Stvari koje se ponavljaju izolirajte u pomoćne metode pa ih pozivajte iz metode `main` koliko puta je potrebno.

Ako se programu preda broj argumenata koji je različit od 0 i različit od 2, program treba ispisati prikladnu poruku i prekinuti s radom.

Upozorenje: ako za čitanje s tipkovnice koristite razred `Scanner`, ali ne njegovu metodu `nextDouble()` jer ista koristi lokalizacijske postavke, već koristite metodu `next()` koja vraća string koji preko razreda `Double` i metoda koje smo spomenuli pokušajte isparsirati u `double`. Time želimo osigurati da korisnik uvijek unosi decimalnu točku. Pri ispisu brojeva također se mora koristiti decimalna točka što statička metoda `Double.toString(...)` poštuje.

Zadatak 3.

U paketu `hr.fer.zemris.java.hw01` napravite program `UniqueNumbers`. Program se pokreće bez argumenata. U programu napravite javnu pomoćnu strukturu `TreeNode` koja ima članske varijable `left` i `right`, te `value` koji je tipa `int` (ne smije imati *ništa* osim toga). Napravite u programu pomoćne metode koje dodaju čvor u uređeno binarno stablo (lijevo manji, desno veći), samo ako čvor s vrijednosti koja se dodaje već ne postoji (u suprotnom, metoda ne radi ništa). Evo primjera uporabe.

```
TreeNode glava = null;
glava = addNode(glava, 42);
glava = addNode(glava, 76);
glava = addNode(glava, 21);
glava = addNode(glava, 76);
glava = addNode(glava, 35);
```

Ako ste to dobro implementirali, stablo će imati četiri čvora i vrijedit će:

```
glava.value: 42
glava.left.value: 21
glava.left.right.value: 35
glava.right.value: 76
```

Napravite metodu koja vraća veličinu stabla. Evo primjera uporabe.

```
int velicina = treeSize(glava);
```

Pozovemo li je nakon prethodnog primjera, rezultat bi morao biti 4. Pozovemo li je prije prvog poziva `addNode`, rezultat bi morao biti 0.

Istestirajte obje metode. U datoteci koja sadrži testove importajte tip `hr.fer.zemris.java.hw01.UniqueNumbers.TreeNode` pa ćete u testovima moći deklarirati varijable izravno po tom "kratkom" imenu.

Napravite metodu `containsValue` koja vraća nalazi li se traženi element u stablu. Primjerice, sljedeće bi nakon prethodnog primjera trebalo ispisati `da`.

```
if(containsValue(glava, 76)) {
  System.out.println("da");
}
```

Napravite tu metodu i istestirajte je. Primijetite, sama metoda `containsValue` ništa ne ispisuje.

Potom napravite glavni program tako da od korisnika s tipkovnice čita broj po broj i dodaje ih u stablo, samo ako tamo već ne postoje (a što se točno dogodilo, ispisuje na zaslon). Za rad sa stablom koristite samo prethodno napisane metode; nemojte pisati novu koja trči po stablu i gleda može li dodati element, pa ako ne može, ispisuje poruku na ekran, a inače dodaje element – to rješavate na drugom mjestu! Korisnik unos prekida utipkavanjem "`kraj`". Jednom kad je unos gotov, program treba ispisati brojeve najprije sortirano od manjeg prema većem, a potom u sljedećem retku od većeg prema manjem. Evo primjer očekivane interakcije.

```
Unesite broj > štefica
'štefica' nije cijeli broj.
Unesite broj > 3.14
'3.14' nije cijeli broj.
Unesite broj > 42
Dodano.
Unesite broj > 76
Dodano.
Unesite broj > 21
Dodano.
Unesite broj > 42
Broj već postoji. Preskačem.
Unesite broj > 35
Dodano.
Unesite broj > kraj
Ispis od najmanjeg: 21 35 42 76
Ispis od najvećeg: 76 42 35 21
```

Ako zbog bolje organizacije koda trebate pomoćne metode, slobodno ih napravite. Kako nije baš jednostavno testirati kod koji čita s tipkovnice odnosno piše na zaslon, pa taj dio niti nemojte testirati. Testirajte samo metode koje ne rade interakciju s korisnikom.

Napomena.

Prilikom rješavanje svih domaćih zadaća, pa tako i ove, zadatke morate rješavati samostalno. Smijete se konzultirati s drugim polaznicima vještine PRIJE no što krenete programirati (što treba riješiti, koja je ideja, kako bi se rješenje moglo oblikovati i slično). Jednom kad krenete programirati, dužni ste sami samostalno napisati rješenje. Ako nešto tada zapne, postoje konzultacije. Gledanje tuđeg rješenja domaće zadaće ili čak uporaba te krađa dijelova koda i ugradnja istoga u vlastitu domaću zadaću smatra se varanjem (bilo da ste nakon toga uhvaćeni ili ne – budite svjesni ako to radite, da to radite) i imat će posljedice. Proučite još jednom ostatak pravila iz uvodne (nulte) prezentacije.

U ovoj prvoj zadaći ne smijete koristiti biblioteke koje nismo obradili; posebice ne smijete koristiti *Java Collection Framework* te gotove implementacije kolekcija. Ako niste sigurni smijete li nešto koristiti, postavite pitanje u Ferku na *Pitanja i odgovori* (imate na stranici kolegija link gore na vrhu ekrana u drugoj traci).

Pročitajte u knjizi dio poglavlja 2 (stranice 21-32) te poglavlje 4 (stranice 75 do 91). Također, prođite još jednom ako je potrebno kroz dokument koji smo koristili na predavanju.

Predaja domaće zadaće.

Eclipse projekt potrebno je zapakirati u arhivu imena `hw01-0000000000.zip` (zamijenite nule Vašim JMBAG-om). Obavezno provjerite da u tu arhivu NE zapakirate skriveni *Eclipseov* direktorij `.settings`, direktorij `target` (ili bilo što što se automatski gradi) kao niti datoteke `.classpath` i `.project` koje stvara *Eclipse*. Sadržaj arhive mora biti valjani Maven projekt (u arhivu se pakira vršni direktorij projekta, poddirektorij `src`, datoteka `pom.xml` i slično). Ovo možete učiniti i izravno iz *Eclipsea*: desni klik na projekt, `Export... → General → Archive File` pa u prozoru koji se otvori odznačite kvačice uz `.classpath` i `.project`, raširite direktorij projekta, odznačite kvačite uz `.settings` i `target`, u opcijama odaberite ZIP te "`Create only selected directories`", i konačno odaberite gdje će se napraviti arhiva i kako će se zvati.

Jednom kad napravite arhivu, zatvorite *Eclipse*, pokrenite ga ponovno i napravite novi pomoćni *workspace*. Provjerite možete li importati napravljeni projekt i to izravno iz ZIP arhive. Potrebni koraci za import u ovom su slučaju sljedeći.

`File → Import... → General → Projects from Folder or Archive`, u prozoru koji se potom otvori klik na gumb "`Archive...`", i odaberite napravljenu ZIP arhivu (na isti ćete način importati domaće zadaće koje ćete trebati recenzirati). Nakon analize arhive, prikazat će se informacija da je pronađen Maven projekt i njemu nadređeni prazan projekt (uz taj maknite kvačicu). Potom klik na gumb `Finish`. Postupak će sam, čim odaberete arhivu, istu ekspandirati u *workspace* direktorij (primjerice, ako je zip arhiva `hw01-0000000000.zip`, napravit će se direktorij `hw01-0000000000.zip_expanded`, i to će postati i naziv projekta u *Project Exploreru*). Ako na disku želite smislenije ime, možete sami u *workspace* direktoriju odkomprimirati arhivu pod "normalnim" imenom (npr. `hw01-0000000000`), i potom napraviti izravno import Maven projekta.

Sav kod mora imati prikladan javadoc – proučite u knjizi taj dio.

Rješenje predajete na Ferka (idete na stranicu kolegija, pa *Komponente*, *Domaće zadaće*, *1. domaća zadaća*, *Upload*). Trebate uploadati ZIP-arhivu; nikako RAR-arhivu, 7z-arhivu ili nešto četvrto. Nakon što ste sigurni da je to konačno rješenje, upload morate zaključati, nakon čega više neće biti moguće raditi novi upload. Predaje koje nisu zaključane, smatraju se nepredanima. Rok za upload i zaključavanje je četvrtak, 14. ožujka, u 7:00:00 ujutro.