Formalne metode u oblikovanju sustava

2. domaća zadaća - Java PathFinder

Zagreb, travanj 2020.

Sadržaj

Izjava

Zadaci priloženi uz ovu domaću zadaću pripadaju njihovim vlasnicima te služe isključivo u edukacijske svrhe. Bilo koja promjena originala je isključivo radi estetske i funkcionalne prirode i ne mijenja smisao informacije. Također, takve promjene ne primijenjuje nikakvu drukčiju licencu niti se smatraju intelektualnim vlasništvom uređivača.

1. DIO

Proučite strukturu projekta `jpf-core`. Primjeri nad kojima će se raditi provjera modela nalaze se u paketu `src/examples`. Osim ako nije drugačije zadano, programi se u **NetBeansu** mogu pokrenuti desnim klikom na odgovarajuću `*.jpf` datoteku i odabirom opcije `Verify...`. Alternativno, ako niste uspjeli podesiti plugin `Verify...` da radi (vidjeti instalacijske upute), moguće je pokrenuti program unoseći puni put u komandnoj liniji koji je obično ovakvog oblika:

```
java -jar >jpf-core lokacija>\build\RunJPF.jar +shell.port=4242 <jpf-core
↪  lokacija>\src\examples\HelloWorld.jpf
```

Zadatak 1.1

Otvorite konfiguracijsku datoteku projekta `jpf-core` koja se naziva `jpf.properties` i koja se nalazi u korijenskom direktoriju projekta.

Koja se defaultna strategija koristi za pretragu prostora stanja u JPF-u? Koja se standardna svojstva provjeravaju prilikom pretrage korištenjem odgovarajućih slušača? Osim naziva strategije i svojstava, navedite i puna kvalificirajuća imena razreda u projektu `jpf-core` koji za to služe.

Odgovor

Defaultna strategija koja se koristi za pretragu prostora stanja je `gov.nasa.jpf.search.DFSearch`.

Standardna svojstva koja se provjeravaju prilikom pretrage su:

- `gov.nasa.jpf.vm.NoUncaughtExceptionsProperty`
- `gov.nasa.jpf.vm.NotDeadlockedProperty`

Zadatak 1.2

Proučite najjednostavniji primjer aplikacije `HelloWorld.java` (u paketu `examples`). Što se ispisuje pokretanjem provjere modela tog programa u dijelu nakon `search started`? Koje su pogreške dojavljene?

Odgovor

Nakon što pokrenemo program, ispisuje se `"I won't say it!"`. Nakon toga dolazi `"no errors detected"`, što nam govori da prilikom testiranja programa nisu pronađeni problemi.

Zadatak 1.3

Proučite primjer ograničenog međuspremnika `BoundedBuffer.java`. Navedite koji su sudionici u ovom primjeru.

Odgovor

Sudionici su proizvođači i potrošači.

Zadatak 1.4

Kojim mehanizmom u Javi su ostvareni ti sudionici? Koje metode koriste koji sudionici?

Odgovor

Sudionici nasljeđuju `Thread` i svaki implementira metodu `run()`. Koriste se metode `get()` i `put()`, koje su sinkronizirane uz pomoć metoda `notify()` i `wait()`.

Zadatak 1.5

Pokrenite aplikaciju ograničenog međuspremnika. Koje svojstvo je narušeno izvođenjem ovog programa koristeći argumente navedene u konfiguracijskoj datoteci (2,4,1)? Što se dogodilo s pojedinim sudionicima? Kolika je bila veličina međuspremnika u ovom slučaju?

Odgovor

Narušavamo svojstvo `gov.nasa.jpf.vm.NotDeadlockedProperty`. U programu smo dali međuspremniku veličinu 2. Međutim, imamo 4 proizvođača i 1 potrošača. Napunit ćemo međuspremnik, ali će sve dretve biti u stanju `WAIT`, što je definicija potpunog zastoja.

Zadatak 1.6

Pokrenite istu aplikaciju, samo s argumentima (4,1,1). Kakva je sad situacija? (Napomena: NetBeans će vas možda gnjaviti da ne može spremiti izmjene u datoteci `*.jpf` jer da je datoteka otvorena vjerojatno samo za čitanje. Obično spremanje promjena ipak uspješno prođe nakon što prođe neko vrijeme (manje od minute), no u slučaju da ne prođe, možete napraviti `Save As...` i pohraniti datoteku pod drugim imenom)

Odgovor

Sada imamo dovoljno mjesta u međuspremniku za normalnu operaciju proizvođača i potrošača, pa ne narušavamo nikakvo svojstvo.

Zadatak 1.7

Opišite ukratko što rade i nad čime se pokreću Javine metode `wait()` i `notify()`.

Odgovor

Aktivna dretva poziva ove metode. Ovisno o pozvanoj metodi, događa se sljedeće:

- `wait()` - dretva se odriče svojeg prava na izvršavanje i čeka dok neka druga dretva ne pozove svoj `notify()`
- `notify` - odabire se jedna dretva, te nakon što trenutna dretva izađe iz kritičnog odsječka se odabranoj dretvi daje mogućnost da uđe u kritični odsječak

Zadatak 1.8

Obrazložite ukratko (i precizno) zašto dolazi do narušavanja svojstva u ovom primjeru.

Odgovor

Nakon što proizvođač stavi nešto u spremnik, on će pozvati `notify()`. Recimo da je proizvođač pozvao `notify()` nakon što je popunio zadnje mjesto u međuspremniku i da je time pozvao drugog proizvođača. Taj proizvođač zove `wait()`, i čeka da netko pozove `notify()`, međutim to je bila jedina aktivna dretva, i sad sve dretve čekaju, tj. dogodio se potpuni zastoj.

Zadatak 1.9

Proučite primjer `Rand.java` i pridruženu konfiguracijsku datoteku `Rand.jpf`. Što specificira konfiguracijska naredba `cg.enumerate_random = true` i zašto je ona bitna za ovaj problem?

Odgovor

Naredba će potaknuti program da provjeri sve mogućnosti kad se u `a` stavi nasumični broj iz {0, 1}, a u `b` stavi nasumičan broj iz {0, 1, 2}. U varijablu c se, potom, stavlja rezultat operacije $\frac{a}{a+b-2}$, pri čemu postoji šansa da se dogodi dijeljenje s 0, što će baciti iznimku jer smo prisilili JPF da provjeri sve mogućnosti postavljanjem `cg.enumerate_random = true`.

Zadatak 1.10

Pokrenite aplikaciju za verifikaciju. Koje svojstvo je ovdje narušeno? Što je programer ovog ili ovome sličnog kôda previdio? Kako se mogao unaprijed osigurati da se cijeli sustav ne sruši? Koje su konkretne vrijednosti varijabli `a` i `b` srušile program?

Odgovor

JPF će provjeriti svojstvo za vrijednosti `a = 0` i `b = 2`, nakon koje će dojaviti

```
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.ArithmeticException: division by zero
```

Programer je previdio da bi se moglo dogoditi dijeljenje s 0, a od toga se lako može osigurati ako se uhvati ta iznimka `try-catch` blokom.

Zadatak 1.11

Sad izbrišite specifikaciju `cg.enumerate_random = true` iz konfiguracijske datoteke te pokrenite aplikaciju. Što se sad dogodilo? Objasnite.

Odgovor

Nije prijavljen problem. Pretpostavljamo da se ovo dogodilo jer JPF nije naletio na ilegalnu konfiguraciju varijabli `a` i `b` jer ih nije sve provjerio.

2. DIO

Zadatak 2.1

Proučite primjer `Racer.java` i konfiguracijsku datoteku `Racer.jpf`. Napravite kopiju konfiguracijske datoteke koju ćete nazvati `Racer_2.jpf` i u kojoj ćete izbrisati liniju `listener=gov.nasa.jpf.listener.PreciseRaceDetector`. Pokrenite aplikaciju za verifikaciju putem `Racer.jpf` i zatim putem `Racer_2.jpf`. Koje svojstvo je narušeno u slučaju `Racer_2.jpf`, a što piše pod `error 1` u slučaju `Racer.jpf`?

Odgovor

Za `Racer_2.jpf` dobivamo ispis

```
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.ArithmeticException: division by zero
```

Za `Racer.jpf` dobivamo `error 1` ispis

```
gov.nasa.jpf.listener.PreciseRaceDetector
race for field Racer@15b.d
```

Zadatak 2.2

Opišite zašto može doći do problema prilikom izvođenja ovog primjera. Može li instanca dretve `t` pristupiti liniji kôda `int c = 420 / racer.d;` ?

Odgovor

Pokretanje dretvi je ovdje zbog nedostatka sinkronizacije proizvoljno. Ako se prvo pokrene dretva koja postavlja `d` na vrijednost 0, onda će doći do dijeljenja nulom.

Instanca dretve `t` ne može pristupiti toj liniji koda jer se ne nalazi unutar vlastite `run()` metode.

Zadatak 2.3

Otvorite kôd razreda `gov.nasa.jpf.listener.PreciseRaceDetector`. Ukratko pojasnite (na temelju komentara razreda) koja je ideja kod implementacije detektora utrke za resursom. Također navedite koji Adapter nasljeđuje ovaj slušač i koje metode nadjačava.

Odgovor

Ideja je prilično jednostavna - izoliraj sve moguće objekte za koje bi se dretve mogle natjecati. Zatim, kako imamo konačan broj dretvi, izvršavamo ih u različitim redosljedima. Ako nakon ovog nemamo problema, pokrili smo sve slučajeve i onda garantirano nema utrke dretvi.

Nasljeđeni Adapter je `PropertyListenerAdapter`, a nadjačava metode `check()`, `choiceGeneratorSet()`, `executeInstruction()` `getErrorMessage()` i `reset()`.

Zadatak 2.4

Proučite kôd primjera `NumericValueCheck.java` i konfiguracijsku datoteku `NumericValueCheck.jpf`. Zatim pokrenite aplikaciju. Koju grešku je dojavio JPF?

Odgovor

JPF dojavljuje grešku

```
gov.nasa.jpf.listener.NumericValueChecker
local variable someVariable out of range: 12345,000000 > 42,000000
```

Zadatak 2.5

Primijetite na koji način je specificirano u konfiguracijskoj datoteci na koju varijablu i na koji način se odnosi provjeravanje raspona numeričkih vrijednosti. Pogledajte sad kôd odgovarajućeg slušača koji implementira provjeravanje raspona vrijednosti. Koje su dvije mogućnosti rada tog slušača (na koje dijelove nekog razreda se može primijeniti)? Navedite i sintaksu tih provjera.

Odgovor

Moguće je provjeravati konretne varijable ili polja - ovo je specificirano identifikatorom nakon `range`. Prema tome, iako je u originalnoj datoteci specificirano

```
range.vars = 1
```

Kad bi htjeli provjeravati raspon polja mogli bismo pisati

```
range.fields = 1
```

Zadatak 2.6

Proučite kôd primjera `TestExample.java` i konfiguracijsku datoteku `TestExample-coverage.jpf`. Korištenjem slušača `CoverageAnalyzer` omogućena je analiza pokrivanja kôda. Pokrenite aplikaciju i promotrite rezultantnu tablicu koju je ispisao `CoverageAnalyzer`. Koji razred je bio bolje pokriven s ispitnim primjerima u metodi main? Koje sučelje je morao implementirati ovaj slušač kako bi izmijenio izgled ispisa izvještaja?

Odgovor

Što se tiče `T1`, pokrivene su sve 3 metode, a kod `T2` pokrivene su samo 2 metode. `T1` je bolje pokriven s ispitnim primjerima.

Kako bi izmijenili izgled ispisa izvještaja moramo implementirati sučelje `PublisherExtension`.

Zadatak 2.7

S obzirom na rezultate ispisa i dani kôd, koja bi to bila metoda `<init>()` koja piše u tablici?

Odgovor

To je najvjerojatnije zadani konstruktor razreda.

Zadatak 2.8

Dodajte u konfiguracijsku datoteku `TestExample-coverage.jpf` pod razrede koje treba uključiti za provjeru dodatno i sam razred `TestExample`, uz postojeće razrede `T1` i `T2`. Spremite datoteku i pokrenite aplikaciju za verifikaciju. Proučite rezultat. Iz samog kôda, očito je da će se proći kroz sve linije metode `main`.

Koje naredbe (linije kôda) analizator pokrivanja preskače kad izvještava da je prošao kroz samo $\frac{3}{8}$ linija kôda metode `main` (osma linija kôda uključuje implicitni `return;`)? Kojom specifikacijom bi isključili ispisivanje provjeravanja pokrivenosti grana u izlaznoj tablici?

Odgovor

Analizator je preskočio `assert` naredbe.

Ako želimo isključiti ispisivanje provjeravanja pokrivenosti grana u izlaznoj tablici možemo koristiti specifikaciju `coverage.show_branches = false`.

3. DIO

Instalirajte projekt (paket) `jpf-aprop` koji služi za provjeru specifičnih svojstava programa pisanih u Javi koja su zadana u kôdu u obliku anotacija (oznaka `@`). Projekt možete naći u zip obliku na stranicama predmeta FMUOS, u materijalima za DZ2. Projekt raspakirajte u isti direktorij gdje se nalazi i projekt `jpf-core` (npr. `NetBeansProjects`). U NetBeansu otvorite novi projekt: odaberite `File` → `New Project` → `Java` → `Java Free-Form Project`. Pod `Location` odaberite direktorij gdje se nalazi `jpf-aprop`. Ostale sve rubrike bi se trebale popuniti automatski. Nastavite dalje. U koraku `Source Package Folder` pod `Source level` izaberite `JDK 1.8`. Zatim odaberite `Finish`. `jpf-aprop` bi se trebao naći u otvorenim projektima. Dodajte u datoteku `site.properties`, koju ste ranije stvorili u postupku instalacije, sljedeće retke na kraj datoteke i pohranite promjene:

```
# annotation properties extension
jpf-aprop = ${jpf.home}/jpf-aprop
extensions+=,${jpf-aprop}
```

Sada pokrenite skriptu `build.xml` koja se nalazi u korijenskom direktoriju projekta jpfaprop (desni klik pa `Run Target` → `build`). Prevođenje datoteka i izgradnja tri `.jar` datoteke trebalo bi proći bez pogrešaka.

Zadatak 3.1

Proučite datoteku `jpf.properties` projekta `jpf-aprop`. Koja se standardna svojstva provjeravaju prilikom pretrage kad se koristi projekt `jpf-aprop`? Gdje je to uopće definirano? Također, navedite put do direktorija s međukôdom razreda koji se kao primjeri provjeravaju uz pomoć `jpf-aprop`. Koje svojstvo pokazuje taj put u datoteci `jpf.properties`?

Odgovor

U datoteci `jpf.properties` nisu definirana nikakva standardna svojstva, znači sve fallbacka na standardna `jpf-core` svojstva.

Relativan put do razrednih direktorija je `src/examples`, a svojstvo koje to pokazuje je `jpf-aprop.sourcepath`.

Zadatak 3.2

Proučite jednostavni primjer `ConstViolation.java` i pridruženu konfiguracijsku datoteku `ConstViolation.jpf` te odgovarajućeg slušača. Opišite što se događa u kôdu razreda `ConstViolation.java`. Koja metoda je označena s `@Const` anotacijom i što to točno znači?

Odgovor

Ono što se događa je da u `main()` metodi se stvara objekt razreda `ConstViolation`, te se nad njim poziva metoda `dontDoThis()`.

Ta ista metoda je označena `@Const` anotacijom - tako označene metode ne smije mijenjati članove svoga razreda, a ako to naprave, izbacuje se `AssertionError`. Ovo svojstvo se delegira i na metode pozvane iz označene metode, tj. ta oznaka otvara tzv. scope u kojem se ne smiju mijenjati članovi razreda.

Zadatak 3.3

Pokrenite tu aplikaciju za verifikaciju. Navedite pogrešku koja je dojavljena.

Odgovor

Dobivamo ispis

```
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.AssertionError: instance field write within const context: int
↪  ConstViolation.d
```

Zadatak 3.4

Proučite primjer `ContractViolation.java` i pridruženu konfiguracijsku datoteku `ContractViolation.jpf`. Navedite koje sve slušače koristi ovaj program (puna kvalificirajuća imena).

Odgovor

Program koristi sljedeće slušače:

- `gov.nasa.hpf.aprop.listener.ConstChecker`
- `gov.nasa.hpf.aprop.listener.ContractVerifier`
- `gov.nasa.hpf.aprop.listener.LockChecker`
- `gov.nasa.hpf.aprop.listener.NonnullChecker`
- `gov.nasa.hpf.aprop.listener.NonSharedChecker`

Zadatak 3.5

Pronađite u strukturi projekta odgovarajućeg slušača u kojem se provjerava svojstvo `nonshared.throw_on_cycle`. U kôdu pronađite i napišite koju bi vrstu iznimke bacio JPF ako bi dretva bila uhvaćena u ciklusu nad objektom koji nije predviđen za višedretvenost.

Odgovor

Radi se o slušaču `gov.nasa.jpf.aprop.listener.NonSharedChecker`.

Kad bi dretva bila uhvaćena u ciklusu nad objektom koji nije predviđen za višedretvenost bacila bi se pogreška

```
java.lang.AssertionError
```

Zadatak 3.6

Što znače anotacije `@Requires`, `@Invariant` i `@Ensures` u kôdu programa `ContractViolation.java`? Koju vrstu dobrog oblikovanja programske potpore ostvaruju ove anotacije?

Odgovor

- `@Requires` definira preduvjete koji bi morali biti ispunjeni prije izvršavanja metode.
- `@Invariant` prisiljava da su tijekom izvršavanja označene metode zadovoljena svojstva invarijante.
- `@Ensures` definira uvjete koji bi morali biti ispunjeni nakon izvršavanja metode.

Ove anotacije ostvaruju programsku potporu zasnivanu na ugovorima (ne znam točno kako se zove jer materijala na internetu nema, a OPP je bio očajan održan što se tiče teorijskog dijela).

Zadatak 3.7

Pokrenite aplikaciju za verifikaciju. Koja anotacija je narušena? Napišite pogrešku koja je dojavljena.

Odgovor

Narušen je dio ugovora označen s `@Ensures`, a dojavljuje se pogreška

```
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.AssertionError: postcondition violated: "(Result >= 0) AND
↪  (Result > 0)", values={Result=0}
```

Zadatak 3.8

Pažljivo proučite dojavljenu pogrešku. Koja metoda kojeg točno razreda je izazvala narušavanje ugovora?

Odgovor

To je učinila `ContractViolation.getLoopCount()`.

Zadatak 3.9

Promijenite ugovore dviju metoda tako da obje ispituju uvjet `(Result>=0)`. Spremite izmijenjenu datoteku `ContractViolation.java`. Pokrenite ponovno skriptu `build.xml`. Kad se izmijenjeni primjer preveo, ponovno ga pokrenite. Kakva je sad situacija?

Odgovor

Sad JPF ne dojavljuje pogreške.

4. DIO

Zadatak 4.1

U NetBeansu napravite novi projekt (`File` → `New Project` → `Java Application`) koji ćete nazvati `JavaFV`. Napravite ga bez razreda `JavaFV` s metodom `main`. Zatim desnim klikom na `Source Packages` napravite novi paket pod nazivom `fv`, a onda desnim klikom na paket `fv` napravite novi razred pod imenom `Verifikacija.java` i statičkom metodom `main` (unutar razreda napišite `public static void main(String[] args))`.

Zadatak 4.1.1

Napravite unutar istoga paketa novu datoteku (desni klik na `fv`, pa `New` → `Other...` → `Other` → `Empty File` i nazovite ga `Verifikacija.jpf`. U tu datoteku dodajte zasad samo jedan redak kojim ćete omogućiti pokretanje razreda `Verifikacija.java` iz paketa `fv` i spremite ju. Kako izgleda taj redak?

Odgovor

```
target = fv.Verifikacija
```

Zadatak 4.1.2

U korijenskom direktoriju projekta `JavaFV` zatim napravite datoteku `jpf.properties` jednostavnog sadržaja:

```
JavaFV=${config_path}
JavaFV.classpath=${JavaFV}/build/classes
JavaFV.sourcepath=${JavaFV}/src/fv
```

Na kartici `Projects` kliknite desnim klikom na vaš projekt `JavaFV` i odaberite `Clean and build`. Nakon što se projekt izgradio, provjerite da se međukôd `Verifikacija.class` nalazi pod direktorijem `build/classes/fv`. Probajte pokrenuti verifikaciju koja bi trebala proći bez pogrešaka. Objasnite zašto je redak `JavaFV.classpath=$JavaFV/build/classes` nužno navesti u datoteci `jpf.properties`?

Odgovor

JPF neće ispravno raditi ako mu ne predamo putanju do međukoda jer ga ne zna otkud učitati.

Zadatak 4.2

Sada izmijenite sadržaj datoteke `Verifikacija.java` tako da sadrži kôd koji se nalazi u istoimenoj datoteci koja se nalazi u repozitoriju kolegija FMUOS (pod DZ2). Također, izmijenit ćete sadržaj datoteke `Verifikacija.jpf` tako da sadrži specifikacije prema istoimenoj datoteci koja se nalazi u repozitoriju kolegija. Nakon kopiranja kôda i specifikacija spremite datoteke, no nećete moći prevesti datoteku `Verifikacija.java` budući da sadrži importe koji su nepoznati projektu `JavaFV`.

Zadatak 4.3

Potrebno je uključiti izgrađene knjižnice (`.jar`) od `jpf-core` i `jpf-aprop` u projekt `JavaFV` kako bi se kôd razreda `Verifikacija.java` mogao prevesti. To se radi tako da odaberete `JavaFV` pa desni klik, a zatim `Properties` → `Libraries` → `Add JAR/Folder`. Pronađite u direktoriju `jpf-core` → `build` → `jpf.jar`, `jpf-classes.jar` i `jpfannotations.jar` te ih dodajte. Ostale knjižnice nisu bitne. Od projekta `jpf-aprop` potrebno je dodati samo knjižnicu `jpf-aprop-annotations.jar`. Odaberite `Ok`. Pogreške bi sada trebale nestati. Sad prevedite `Verifikacija.java` (`desni klik` → `Compile File`).

Zadatak 4.4

Pokrenite aplikaciju za verifikaciju. Koja pogreška vam je dojavljena? Objasnite zašto je došlo do te pogreške s obzirom na konfiguracijsku datoteku i zadani kôd.

Odgovor

Javlja se pogreška

```
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.AssertionError: null assignment to @Nonnull instance field:
↪  Verifikacija.id
```

Pregledom izvornog kôda, vidimo da `v.setId(null);` narušava specifikaciju `@Nonnull private String id;`.

Zadatak 4.5

U konfiguracijsku datoteku dodajte ovaj redak na kraj:

```
search.multiple_errors = true
```

Ovime se prolazi svim putevima izvođenja kroz program i dojavljuje se za svaki put izvođenja prva pogreška na koju se naletilo. Pokrenite aplikaciju za verifikaciju. Koja je razlika između prethodnog ispisa pogrešaka i sadašnjega?

Odgovor

Sad imamo dvije dodatne pogreške, obje jednake:

```
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.AssertionError: null assignment to @Nonnull instance field:
↪  Verifikacija.id
```

Zadatak 4.6

Uklonite redak `search.multiple_errors = true` te zakomentirajte redak u kôdu koji smatrate odgovornim za dojavu pogreške iz **zadatka 4.4**. Prevedite kôd i pokrenite ponovno aplikaciju za verifikaciju. Koja se sada pogreška pojavila, na kojem retku kôda i zašto je prijavljena?

Odgovor

Sad se pojavilo sljedeće:

```
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.AssertionError: write of const instance field outside
↪  constructor: java.lang.String Verifikacija.DNA
```

Javlja se pogreška jer smo pokušali izmijeniti varijablu `DNA` koju smo označili prethodno s `@Const`. Ovo se događa u liniji

```
v.setDNA("CTGA");
```

Zadatak 4.7

Zakomentirajte redak u kôdu koji smatrate odgovornim za dojavu ove vrste pogreške. Prevedite kôd i pokrenite ponovno aplikaciju za verifikaciju. Koja je sada pogreška dojavljena? Objasnite zašto se ova pogreška dogodila.

Odgovor

Sad se javlja pogreška

```
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.AssertionError
```

Ovo se dogodilo jer smo u `.jpf` datoteci definirali raspon varijable kao [20, 100]. Međutim, imamo sljedeću liniju:

```
assert (vel < 100.0);
```

Ovo će pasti kad nam `vel` bude jednak 100 jer ne vrijedi $100 < 100$.

Zadatak 4.8

Izmijenite naredbu (`assert`) u kôdu tako da više ne dolazi do ove vrste pogreške. Prevedite kôd i provjerite da se pogreška zaista više ne događa. Koji ste broj trebali navesti kao uvjet u naredbi (`assert`) da ne dođe do pogreške?

Odgovor

Izmjena sporne linije u

```
assert(vel <= 100)
```

će se riješiti pogrešaka prilikom verifikacije. Ako ne smijemo mijenjati znak, onda kao zamjenski broj možemo umetnuti npr. 100.1.

Zadatak 4.9

Provjerite metode razreda `gov.nasa.jpf.vm.Verify`. Postoje li metode `getDouble()` i `getInt()` bez argumenata? Objasnite. Objasnite na primjeru što su to generatori izbora i koja je namjena navođenja heuristika pri korištenju generatora izbora. Koja se heuristika koristila u primjeru `Verifikacija.java`?

Odgovor

Metode `getDouble()` i `getInt()` bez argumenata ne postoje. Nema previše smisla pozivati ove metode bez nekog zadanog raspona ili referencom na ključ definiran u `.jpf` datoteci.

Generatori izbora su ponašanje u JPF uz pomoć kojega ne moramo pretraživati sva moguća stanja, već samo ona bitna. Dakle, radi se o nekakvoj heuristici. Umjesto da provjeravamo sva moguća stanja, njih je previše, ispitati ćemo samo relevantna stanja. Heuristika treba biti takva da njena manjkavost u broju provjerenih stanja ne utječe na ispravnost algoritma.