

Debugovanje Java Programa

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Ivan Vrankovic, Andjela Karakas, Ivan Mihajlovic, Kristina Petrovic

3. april 2020.

Abstract

Tema ovog rada je debugovanje Java programa. U nastavku ćemo se baviti detekcijom bagova, tehnikama otklanjanja i alatima za tu namenu, kao i opštim savetima pri debugovanju. Obuhvat ćemo sve vrste debagera i najpoznatije predstavnike. Tu će se izdvojiti Java debager, debageri integrisanih okruženja Eclipse i NetBeans, predstavnik debagera istorije Chronon DVR i Replay Solutions, alat BTrace za dinamičko praćenje, alati za produkciono debugovanje. Ako navedeni debageri ne zadovoljavaju Vaše potrebe, razmotrite pravljenje sopstvenog uz pomoć visenitne debugujuće arhitekture - JDPA. U nastavku će akcent biti na serverskoj strani debugovanja na kojoj više dolaze do izražaja specifičnosti Java debugovanja. Saznaćete kako se debuguju servleti, JavaServer stranice i Enterprise JavaBeans komponente. Na kraju ćemo demonstrirati teorijski deo rada na praktičnom primeru.

Contents

1	Java debugovanje	2
1.1	Deklarativno Java debugovanje	2
1.2	Debugovanje konkurentnih Java programa	2
1.3	Testiranje na različitim platformama	3
2	Java debager	3
3	Kategorije i tipovi grešaka	3
4	Tipovi debagera i njihovi predstavnici u Javi	3
4.1	Debageri u interfejsu komandne linije	4
4.1.1	Tačke prekida	4
4.1.2	Koraćanje(eng.Stepping)	5
4.1.3	Izuzeci	5
4.2	Debageri integrisani u razvojno okruženje	5
4.3	Samostalno napravljeni debageri	5
4.4	Skladište hipa	6
4.5	Debageri istorije	7
4.6	Dinamičko praćenje	7
4.7	Produkciono debugovanje	7
5	Debugovanje serverskih aplikacija	8
5.1	Debugovanje servleta	8
5.2	Debugovanje JavaServer Stranica(eng.JavaServer Pages, JSP)	8
5.3	Debugovanje Enterprise JavaBeans (EJB)	9
6	Debugovanje čestih grešaka Java programa	9
6.1	Debugovanje grešaka prevodioca	9
6.2	Debugovanje Run-Time grešaka	10

1 Java debugovanje

Debugovanje (eng. debugging) je proces otklanjanja svega što stvara probleme u funkcionisanju softvera. Proces razvoja softvera ne može proći bez pojave бага. Багове разликујемо по сложености на синтаксне, логичке, рантајм (eng. run-time) грешке и оне најтеже за оtklanjanje - нитне грешке (eng. threading errors). Први Java програмери користили су методу debugovanja коришћењем System.out.println() метода. Данас, ово има смисла користити код мањих делова кода који не захтевају детаљну анализу debagerom, која троши време и меморију. Са порастом сложености система јавила се потреба за алатима који би штедели на времену детекције багова. Ради додатне сигурности језик је проширен elementима као што су: сакупљач отпадаци (eng. garbage collection), провере граница низова, интерфејси и пакети.

1.1 Deklarativno Java debugovanje

Deklarativan debager (eng. declarative debugger), nasuprot tradicionalnom debageru koji ne koristi prednosti povezanosti konponenta jezika kao sto je Java, nudi fleksibilniji pristup debugovanju, na višem nivou apstrakcije. Umesto ispitivanja svake instrukcije nakon tačke prekida (eng. break point), koristi se sekvenca pitanja za redukovanje prostora pretrage.

Sa prvim neočekivanim ponašanjem počinje se sa kreiranjem stabla odlučivanja (eng. computation tree). Uloga testera je da odgovara na pitanja pri obilasku stabla. Debager će iskoristiti tu informaciju da isključujući iz razmatranja sve naslednike čvora koji ne ispunjava korisnikov kriterijum. Svaki čvor odgovara rezultatu nekog podizracunavanja i pridružen mu je odgovarajući deo koda kojim se taj rezultat dobija. Rezultati svih naslednika roditeljskog čvora (eng. parent node) su neophodni da bi se dobio roditeljski rezultat. Koreni čvor odgovara rezultatu glavnog izracunavanja. Kvarnim čvorom (eng. buggy node) se smatra čvor koji daje pogresan rezultat od ispravnih rezultata njegovih naslednika. On ukazuje na neispravan metod čije informacije sadrži. U nekoliko eksperimenata pokazano je da deklarativno debugovanje daje dobre rezultate i relativno brzo nalazi greske. U Javi je moguće implementirati ovakav tip debugovanja kroz JPDA (Java Platform Debugging Architecture).

1.2 Debugovanje konkurentnih Java programa

Nepredvidiv redosled izvršavanja niti konkurentnih programa otežava ponavljanje uslova u kojima se ispoljila greška, a time i njeno razumevanje i otklanjanje. Rezultati interakcija niti takodje mogu biti neočekivani. Ponovno izvršavanje programa i analiza trenutnog stanja programa, u ovom slučaju, nije od pomoći i usporava proces debugovanja.

Java nudi mehanizme za izbegavanje ovakvih vrsta grešaka kroz sinhronizaciju niti. Metod synchronized() aktivira mehanizam pristupa preko brave (eng. lock) nad zajedničkim objektom. On se ogleda u uzajamnom isključivanju niti čime se obezbeđuje ekskluzivan pristup deljenim podacima i njihov integritet.

Različite verzije JVM različito definišu prioritet niti, pa se save-tuje testiranje programa na različitim platformama. Jos jedan način na koji se povećava šansa neuspeha, kod testiranja, je nasumična i česta razmena niti.

Sinhronizacija se ostvaruje kroz blokiranje odredjenih procesa, samim tim utiče na performanse programa. Zato pažljivo razmotriti segmente koda koje je neophodno zaštititi.

1.3 Testiranje na različitim platformama

Mnogi programeri koriste tehniku vezivanja debagera za aktivan proces (eng. Attaching to a running process) u modu za debugovanje ili aktivnu JVM, u lokalnom ili udaljenom (eng. remote) sistemu, za rešavanje problema kočenja programa u nekim određenim vremenskim intervalima koja mogu biti izazvana beskonačnim petljama.

Udaljeno debugovanje (eng. remote debugging) je proces izvršavanja aplikacije na jednom sistemu i njeno debugovanje na drugom. Udaljeno debugovanje je naročito korisno za složene GUI aplikacije. Neki debageri podržavaju ovakav vid debugovanja.

2 Java debager

Debugovanje Jave je kompleksan pojam. Zbog toga postoji mnogo tipova debagera i mnogo alata koji to omogućavaju, a na nama je koji ćemo izabrati. Naravno, izbor debagera zavisi i od situacije. U ovom odeljku ćemo prikazati razlike među vrstama debagera i prikazati glavne predstavnike svake kategorije koji će vam pomoći da odaberete odgovarajući alat za odgovarajući posao.

3 Kategorije i tipovi grešaka

”Bagovi” su greške u programu koje dovode do nekorektnog ili neočekivanog stanja programa, ili čine da program ispoljava neočekivano ponašanje. Različiti efekti ovoga se mogu grubo kategorisati u:

1. Neočekivana kontrola toka (eng. Unexpected flow control)
2. Neočekivane alokacije hipa (eng. Unexpected heap allocations)
3. Odložena kontrola toka (eng. Delayed flow control)

Neočekivana kontrola toka dovodi do izuzetka ili do lokacije, tj. stanja programa u kojem nismo želeli da se nadjemo. Ovde su debageri najčešće korišćeni - da se otkrije korelacija koda i stanja.

U slučaju neočekivanih alokacija hipa alociraju se ili preveliki objekti ili previše manjih objekata. Zadržavanje dugotrajnih referenci na njih čini situaciju još gorom. U ovome analize hipa imaju veliku ulogu. Te analize utvrjuju da li postoje reference na vrednosti koje ”žive” i van funkcije u kojoj su te vrednosti deklarirane.

Odložena kontrola toka se najčešće javlja u slučaju prosleivanja pogrešnog ulaza ka eksternom pozivu (npr. ”SELECT * FROM everything”) ili zaglavljanja u dugu ili beskonačnu petlju. Ovo su situacije kada profajleri performansi uskaču u igru.

Postoje preklapanja među funkcionalnostima alata koji su napravljeni za korišćenje primarno u različitim kategorijama. To je posledica toga što svi oni služe istoj svrsi - prikazivanja stanja programa koje nismo očekivali i popravljanja koda tako da dovodi do očekivanih stanja.

4 Tipovi debagera i njihovi predstavnici u Javi

Tipovi debagera koje ćemo obraditi:

1. Debageri u interfejsu komandne linije (eng. CLI (Command-line interface) debuggers)

2. Debageri integrisani u razvojno okruženje (eng. IDE (Integrated development environment) debuggers)
3. Samostalno napravljeni debageri (eng. Build your own debugger)
4. Skladište hipa (eng. Heap dumps)
5. Debageri istorije (eng. Historical debuggers)
6. Dinamičko praćenje (eng. Dynamic tracing)
7. Produkciono debugovanje (eng. Production debugging)

4.1 Debageri u interfejsu komandne linije

Glavna uloga u ovoj kategoriji pripada jdb (eng. Java debugger) koji dolazi u sklopu Java razvojnog okruženja JDK (eng. Java development kit) koji je u Java virtuelnoj mašini (eng. Java virtual machine, JVM) ekvivalent GDB-u (eng. GNU debugger). jdb koristi interfejs komandne linije preko koje se može povezati sa pokrenutom JVM. Baš kao i GDB, njegova funkcionalnost je robusna i dijapazon funkcionalnosti koje postoje u jdb-u je gotovo identičan funkcionalnostima koje postoje i u debagerima koji su integrisani u razvojnom okruženju. jdb ima u sebi i dodatni alat koji se zove jstack. On omogućava ispisivanje stanja steka u željenom trenutku za svaku nit koja je startovala u pokrenutom programu. Ovo se, međutim, ne odnosi na stanje hipa.

Postoje dva različita načina za pokretanje jdb sesije:

1) Pokretanje jdb sesije dodavanjem klase: pokreće se komandom "`\>jdb ime klase`". Ovo pokreće rad nove virtualne mašine koja učita klasu i zaustavi je pre prve linije u klasi.

2) Dodavanjem jdb pokrenutoj virtualnoj mašini za pokretanje sesije: pokreće se komandom

`"\>java -agentlib:jdpw=transport=dt_shmem,address=jdbconn,server=y,suspend=n ime klase"` sada možemo da povežemo jdb sa virtualnom mašinom koristeći komandu `\>jdb \-attach jdbconn`.

Neke od komandi jdb-a: `run,cont,print,dump,threads,where`.

Komanda	Opis
help ili ?	Ispisuje listu JDB komandi sa kratkim opisom
run	Nakon pokretanja JDB i postavljanja tački prekida, ovom komandom se izvršava program do prve tačke prekida
cont	Nastavlja izvršavanje nakon tačke prekida, izuzetka ili koraka
dump	Za primitivne vrednosti ova komanda je identična običnom ispisu. Za objekte ispisuje vrednosti svih polja definisanih u objektu
print	Prikazuje objekte i primitivne vrednosti
threads	Izlista niti koje su pokrenute
where	Ispisuje sadržaj steka trenutne niti

4.1.1 Tačke prekida

`stop at ime_klase:37` –postavlja tačku prekida na 37 liniji fajla koji sadrži ime_klase `stop in ime_klase.ime_metoda |ime_promenljive` - postavi tačku prekida na odgovarajući metod ili promenljivu unutar klase ime_klase

Ukoliko imamo polimorfizam, moramo da navedemo i tipove argumenta da bi smo postavili tačku prekida na pravi metod. Na primer `ime_klase.ime_metoda (double, double)` ili `ime_klase.ime_metoda(int)`. Komandom `clear` možemo da uklonimo tačku prekida. Na primer `clear ime_klase:37` će ukloniti tačku prekida koju smo postavili na 37 linij.

4.1.2 Koraćanje(eng.Stepping)

U procesu koraćanja se koriste komande step, list i cont.Komanda step nas pomera na sledeću liniju, komanda list nam pokazuje na kojoj se liniji trenutno nalazimo.

4.1.3 Izuzeci

Kada se desi izuzetak za koga ne postoji catch izraz virtualna mašina ispiše izuzetak i završi sa izvršavanjem programa. Kada je pokrenut JDB, izuzetak se ispisuje i pomocu JDB-a se ustanovljava uzrok izuzetka.

Prednosti:

Najveća prednost jdb-a je portabilnost. Može se pokrenuti na serveru veoma brzo bez potrebe za skladištenjem samog debagera na server. Ako se nadjete u nezgodnoj situaciji na serveru i niste u mogućnosti da zaustavite JVM, jdb vam je najbolja opcija za rešavanje tog probl

Nedostaci:

Mana jdb-a, kao i svih drugih alata koji se koriste preko komandne linije, je ta što je možda previše kompleksan za svakodnevno korišćenje. To nas dovodi do sledeće kategorije.

4.2 Debageri integrisani u razvojno okruženje

Debageri integrisani direktno u razvojnom okruženju su, zbog jednostavnosti upotrebe, brz način za debugovanje programa direktno iz razvojnog okruženja. Dva alata prednjače u ovom trenutku. To su Eclipse i NetBeans. Oba debagera koriste sličnu tehnologiju kao i jdb tako što se ili prikače na već postojeću JVM ili pokrenu novu JVM. Iako ne tako prenosivi, funkcionalnosti i mogućnosti koje oni pružaju mogu drastično skratiti proces debugovanja, bez neke velike muke.

Nedostaci:

Kao i sve druge desktop aplikacije koje koriste visok nivo apstrakcije nisu nešto što bi pokrenuli u toku produkcije. Postoji uvek mogućnost debugovanja na daljinu, ali šanse za to u kompleksnim okruženjima su minimalne.

4.3 Samostalno napravljeni debageri

Java Platform Debugger Architecture(JDPA) je višenitna debugujuća arhitektura koja omogućava programerima da kreiraju aplikacije koje rade na virtuelnim mašinama i različitim verzijama JDK-a. Sastoji se iz tri sloja:

1)JVM TI - interfejs za alate java virtuelne mašine(eng. Java Virtual Machine Tool Interface) definiše servise debugovanja koje pruža virtuelna mašina. Ti servisi su zahtevi za informacije (npr. trenutni stek okvir), akcije (npr. postavljanje tačke prekida) i obaveštenja (npr. stigli smo do tačke prekida). Ovo nisu jedine informacije virtuelne mašine koje debager koristi, ali su izvor svih informacija koje pruža debager.

2)JDWP - žičani protokol za debugovanje (eng. Java debug Wire Protocol) definiše komunikaciju između front-enda debagera i procesa koji se debuguje. Protokol nam omogućava da debager i program koji se debuguje rade na različitim virtuelnim mašinama pa čak i na različitim platformama. Koristi informacije koje pruža JVM TI i dodatne informacije neophodne za probleme protoka.

3)JDI - interfejs za Java debugovanje(eng. Java Debug Interface) definiše interfejs visokog nivoa čije alate programeri mogu da koriste za pisanje aplikacija za debugovanje na daljinu.

Komponente:

- proces koji se debuguje(eng.debuggee): sastavljen od aplikacije koja se debuguje, virtuelne mašine u kojoj je aplikacija pokrenuta i back-end debagera
- Java virtuelna masina: označava virtuelnu mašinu na kojoj je pokrenut program koji se debuguje
- back-end: back-end debagera je odgovoran za slanje zahteva front-enda debagera ka virtuelnoj mašini na kojoj se nalazi program i slanje odgovora na te zahtave front-endu. Back-end i front-end komuniciraju preko komunikacionog kanala koristeći žičani protokol za debugovanje (JDWP). Back-end i virtuelna mašina na kojoj se nalazi program komuniciraju preko interfejsa za alate Java virtuelne mašine(JVM TI)
- komunikacioni kanal: je veza izmeu front-end i back-end debagera
- front-end: implementira interfejs za Java debugovanje
- korisnički interfejs: ono što korisnik može da vidi

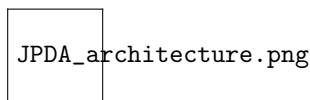


Figure 1: JDPA

Predstavnik:

JSwat je primer samostalno napravljenog debagera korišćenjem ovog alata.

Nedostaci:

Pravljenje svog debagera je ipak veoma kompleksna stvar, pogotovo kada ne želite da promenite stanje programa na koji želite da ga pokrenete. Zbog toga je potreban veoma jak razlog zbog koga se ne mogu savladati problemi već postojeće rasprostranjene alate koji su ipak temeljno testirani.

4.4 Skladište hipa

Skladište hipa je slika memorije Java procesa u određenom trenutku. Ta slika sadrži informacije koje se tiču Java objekata i klasa u trenutku kad je slika napravljena. U nekim situacijama se pokušava popravljanje "mrtvog" programa. U tom slučaju se više informacija može dobiti od slike hipa nego od slike zaustavljenog programa. jmap, koji dolazi sa JDK-om, omogućava generisanje tih slika od pokrenutog programa. Postoji mnogo alata za istraživanje i analizu skladišta. jhat i visualVM, koji takođe dolaze sa JDK-om, su jedni od alata koji su korišćeni u te svrhe. Eclipse-ov dodatak MAT i NetBeans-ov HeapWalker su takođe odlični izbori jer koriste snažno IDE korisničko okruženje.

Situacije u kojima se koriste:

Koristi se kada se javi kompleksna greška i normalne tehnike debugovanja se ne mogu primeniti (npr. aplikacija je pokrenuta na klijentskom serveru). Sa druge strane, skladište hipa se može koristiti i za analizu curenja memorije uključivanjem flega -HeapDumpOnOutOfMemoryError koji čini da se hip automatski prazni kada se napuni.

Nedostaci:

Najveći nedostatak skladišta hipa je taj što obično zauzima jednaku veličinu memorije kao i sam hip, što znači da su u pitanju gigabajti. Nakon toga se skladište hipa šalje nazad korisniku na obradu. Njegovo korišćenje u produkciji takoe nije jednostavno.

4.5 Debageri istorije

Ova kategorija alata nalazi primenu u situacijama kada ne želite da zaustavite JVM ili to niste u mogućnosti. Chronon DVR je dobar primer alata sa takvim pristupom. Ovde debager koristi bajt kod za logovanje podataka. To obično uključuje stvari kao što su naredbe poziva metoda zajedno sa prosleenim parametrima. Ovo omogućava debageru da reprodukuje kod i korisniku pruži uvid u sam tok programa u toku izvršavanja. Replay Solutions je primer sa drugačijim pristupom gde su IO ulazi koji su korišćeni u toku Java programa snimljeni, a zatim reprodukovani. To pruža simulaciju toka programa nakon izvršavanja.

Situacije u kojima se koriste:

Primarna oblast korišćenja ove klase alata je u toku QA-a gde mogu pomoći da se greške reprodukuju tako što će snimiti stanje izvršavanja. Drugi scenario bi bio taj da korisnik pokreće alat ad-hok da bi snimio stanje JVM kada aplikacija krene da ispoljava neočekivano ponašanje u toku produkcije.

Nedostaci:

Najveća mana ovih alata je ta što logovanje ima cenu, a logovanje svega košta mnogo. To znači da usporenje programa korišćenjem ovih debagera može biti od marginalnih, pa čak do 50% što je razlog zašto ovi debageri praktično ne nalaze primenu u produkciji. To drastično ograničava njegovu upotrebu.

4.6 Dinamičko praćenje

Alati u ovoj kategoriji omogućavaju selektivno ispisivanje "putanje" odnosno informacije o stanju bez zaustavljanja programa koji je u toku i bez snimanja svega od pokretanja programa. Zamislite to kao dinamičko umetanje koda koji ispisuje vrednosti iz samog koda koji ćete moći videti. Istaknuti alat ovde je BTrace koji uvodi sopstvenu sintaksu kako bi omogućio korisniku da definiše gde i šta treba pratiti u kodu. Sintaksa je dizajnirana tako da podržava samo read-only operacije kako bi se sprečilo menjanje stanja programa ili da prouzrokuje beskonačnu petlju.

Situacije u kojima se koriste:

Najčešće se koristi kada pokušavate da otklonite grešku na serveru za neki specifični problem (npr. iscrpljivanje skupa veza u bazi podataka (eng. DB connection pool is depleted)) ili želite da prikupite odrene statistike ad-hok bez prekida programa.

Nedostaci:

Baš kao i kod debagera, dinamičko praćenje sa produkcionog servera obično nije preporučljivo (a uglavnom nije ni dozvoljeno).

4.7 Produkciono debugovanje

Glavni alati su frejmvorci za logovanje (log4j, Logback) za logovanje stanja i log analizatori (Logstash, Splunk) za skalirani prikaz

podataka.

Situacije u kojima se koriste:

Kada se bavite produkcionim sistemima zaustavljanje JVM-a da biste pogledali stanje ili skladište hipa se nikako ne radi. To je zato što na taj način praktično isključujete server da ga debugujete, a to se čini samo u ekstremnim situacijama. Metod kojim obično "izvlačimo" stanje iz JVM-a za vreme izvršavanja bez zaustavljanja je selektivnim evidentiranjem promenljivih u log fajl, uglavnom uz pomoć Java frejmworka za logovanje. Kasnije se mogu koristiti razni alati za parsiranje podataka počevši od nečeg jednostavnog kao što je tail, pa sve do skalabilnih log analizatora poput Logstash-a i Splunk-a.

Nedostaci:

Najveći hendikep ovde je taj što moramo unapred znati šta treba logovati i činiti to efikasno. Logovi se, u slučaju nepažljivih programera, mogu vrlo brzo napuniti nebitnim podacima ili se mogu propustiti neki ključni podaci.

5 Debugovanje serverskih aplikacija

5.1 Debugovanje servleta

Servleti su proizvođači servera i rade u višenitnom okruženju na veb serveru, u kome se odvija veliki broj interakcija između servera i korisnika. Samim tim postoji velika šansa za nastanak grešaka koje su veoma teške za otkrivanje. Servlet se ne može direktno debugovati jer radi unutar servera. Za debugovanje nam je potreban server koji izvršava HTTP zahteve. Sun pruža pokretač servleta (eng.Servletrunner) server koji možemo da upotrebimo. On obradjuje samo HTTP zahteve za servlet i ništa više osim toga. Pri izvršavanju pokretača servleta u classpath-u će ubacuju odgovarajuće klase i izvršava se klasa sun.servlet.http.HTTPServer(). Da bi smo debugovali servlet, mi možemo da debugujemo sun.servlet.http.HTTPServer(), HTTPServer() izvršava sve servlete kao odgovor na zahtev korisnika. Sada možemo da postavimo tačke prekida na servlete koje želimo da debugujemo, a za slanje zahteva koristimo veb pretraživač. Prilikom pokretanja servleti se instanciraju samo jednom, nakon toga se pravi posebna nit za svaki korisnički zahtev. Ovde nastaje najveći broj grešaka.

Pored standardnog debagera za otkrivanje grešaka u servletima možemo koristiti logove grešaka (eng.Error log) i logove događaja(eng.Event log). Mnogi serveri ispisuju log grešaka, gde se nalazi lista grešaka u serveru i log događaja, gde se nalazi lista događaja servleta (eng.Servlet events). Korišćenjem log() metoda mozemo da ubacimo dodatne informacije u logove servera.

5.2 Debugovanje JavaServer Stranica(eng.JavaServer Pages, JSP)

Debugovanje JSP fajlova je malo komplikovanije jer se JSP faza kompilacije odvija odmah pre run time-a. Zbog toga se propuštaju sintaksičke greške. Dodatna poteškoća je to što JSP barata sa više slojeva kodiranja tj. JSP se prvo prebacuju u servlete, koji se zatim kompilira u .class fajl koji zatim generiše JavaScript ili HTML kod. Svaki od slojeva može da izazove različite tipove grešaka:

1)Sloj prebacivanja JSP-a u servlet: Greške koje se pojavljuju u ovom sloju su uglavnom sintaksičke greške u JSP kodu. Ove greške

se lako ispravljaju samim gledanjem koda.

2)Sloj kompilacije servleta u .class fajl. Tokom ove faze java kompajler prijavljuje bilo kakvu grešku koju tražimo pomoći standardnog debagera.

3)Poslednji sloj pokreće servlet. JSP engine stavlja ceo servlet kod u jedan veliki try/catch blok koji izbacuje izuzetak(eng. Exception). Mana ovoga je to sto će svi izuzeci biti uhvaceni u RuntimeException() superklasi pa zbog toga ne možemo da utvrdimo specifičnosti nastale greške. Jedan od načina da se ovo zaobidje jeste korišćenje out.flush() da bi se poslao ispis pretraživacu.

Zato što su JSP fajlovi kompilirani u servletima mogu da se pojave višenitne greške kao i kod servleta. JSP nudi mogućnost SingleThreadModel, kojim se ove greške mogu izbeći.

5.3 Debugovanje Enterprise JavaBeans (EJB)

EJB komponente su teške za debugovanje, razlog je taj što se one izvršavaju u kontejnerima (eng.Container), prisiljavajući nas da u debager učitamo i kontejner. Problem je što nisu svi kontejneri napisani u Java kodu, pa ih je nemoguće učitati u debager. Najčešći problemi koji se ovde javljaju jesu korisničke greške i klase generisane od strane EJB kontejnera nisu na istom nivou kao EJB specifikacija. EJB kontejneri imaju okruženja za debugovanje koja omogućavaju da se prodje kroz kod. Kao i kod servleta, EJB imaju log fajlove koji nam mogu pomoći u otkrivanju grešaka.

6 Debugovanje čestih grešaka Java programa

6.1 Debugovanje grešaka prevodioca

Do greške prevodioca dolazi kada prevodilac ne može da razume programsku naredbu. Neki od primera takvih grešaka su pogrešno napisane ključne reči, razmak, nedostajanje interpunkcijskog znaka. Najbolji način za izbegavanje ovakvih grešaka jeste petantan i detaljno ispisani kod, upoznatost sa specifičnostima Java jezika. Neki od primera grešaka prevodioca su:

doubel height;	Problem pogrešne upotrebe ključne reči double.
double height	Problem ne navoenja znaka ; na kraju naredbe.
Double height;	Problem korišćenja velikog slova za naziv tipa promenljive.
public class Hello World	Problem korišćenja razmaka u nazivu klase.

Tipičan primer Java dijagnostike problema:

PrimerKlase.java:15: cannot find symbol

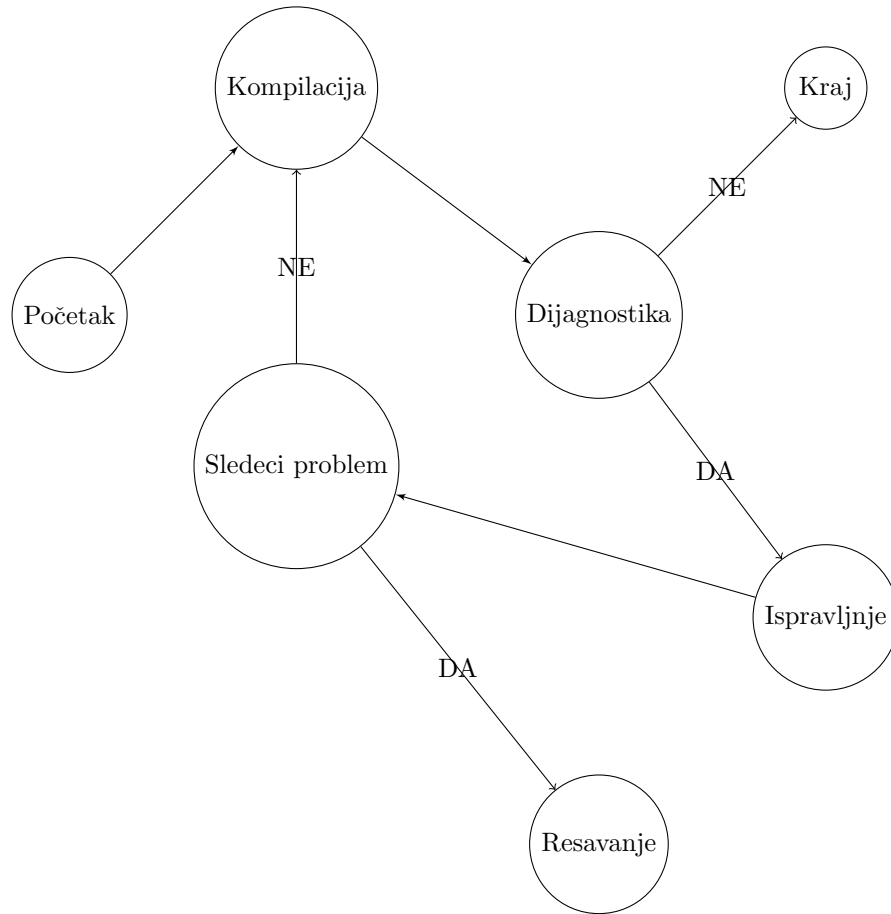
symbol : variable Varijabla

location: class PrimerKlase

Varijabla = 123;

Prva linija govori i nazivu java fajla u komje došlo do greške, druga linija govori o tome okom simbolu se radi, teća linija govori o tome u kojoj klasi je došlo do problema a četvrta linija dijagnostike upotreba simbola.

Koraci procesa dijagnostikovanja greške:



Preporučeno je da se ispravi prva greška i ponovo kompajlira da bi se došlo do preostalih grešaka koje imaju smisla. Razlog da to je zato što preostala dijagnostika može biti lažna dijagnostika, može da identifikuje ispravnu izjavu kao netačnu. Iz tog razloga prevodilac daje lažnu dijagnostiku, program se čita sa levo desno, odozgo prema dole i stvara dijagnostiku ako naiđe na nešto neočekivano. Posle ove prve greške, prevodilac pokušava da se oporavi preskakajući tekst programa dok se nepoznati simbol ne razjasni. Ako preskočeni tekst programa sadrži informacije koje prevodilac mora da analizira u sledećim izjavama, te izjave će dijagnostikovati iako su tačne.

6.2 Debagovanje Run-Time grešaka

Run-time greške ne registruje prevodilac ali može uzrokovati da računar abnormalno prekine sa radom u toku izvršavanja programa.

Primer:

```

java.util.Scanner in = new java.util.Scanner( System.in );
System.out.print( "Upišite broj: " );
double radius = in.nextDouble( );

```

Ukoliko program u izvršavanju dođe do run-time greška, program se prekida i generiše se dijagnostika.

Primer dijagnostike: Exception in thread "main" java.util.InputMismatchException
 at java.util.Scanner.throwFor(Scanner.java:919)
 at java.util.Scanner.next(Scanner.java:1540)

at java.util.Scanner.nextDouble(Scanner.java:2556)
at MPG.main(MPG.java:15)

Redovi vam tačno govore gde je bačen izuzetak i kako se to prenosi kroz lanac poziva metoda.

Java Servlet Programing - Jason Hunter, William Crawford
Beginning JavaServer Pages - Vivek Chopra, Sing Li, Rupert Jones, Jon Eaves, John T. Bell
The Pragmatics Of Java Debugging - Joe Winchester
<https://docs.oracle.com/>
<https://www6.software.ibm.com/developerworks/education/j/j.pdf>
<https://blog.overops.com/java-debugger-the-definitive-list-of-tools/>
<http://faculty.salina.k-state.edu/tmertz/Java/202debugging/01debuggingajavaprogram.pdf>
<https://stackoverflow.com/questions/18075952/db-connection-pool-getting-exhausted-java>
<https://stackoverflow.com/questions/426569/why-is-debugging-better-in-an-ide>
<https://stackoverflow.com/questions/49203089/golang-doing-unexpected-heap-memory-allocation>