Razvoj QUIC protokola

Vrkljan, Krunoslav

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, School of Applied Mathematics and Informatics / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet primijenjene matematike i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:126:698300

Sveučilište Josipa Jurja Strossmayera u Osijeku

fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

Razvoj QUIC protokola

Završni rad

Kandidat:

Krunoslav Vrkljan

Osijek, 2023

Sveučilište Josipa Jurja Strossmayera u Osijeku

fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

Razvoj QUIC protokola

Završni rad

Mentor:

izv.prof.dr.sc. Domagoj Matijević

Komentor:

dr.sc. Mateja Đumić

Kandidat:

Krunoslav Vrkljan

Osijek, 2023

Sadržaj

Sažetak

QUIC je protokol baziran na UDP protokolu koji ima značajke TCP protokola. Pri komunikaciji poslužitelja i korisnika koristi razne pakete ovisno o poruci koju želi poslati. Paketi su zaštićeni enkripcijom tako da ako netko presretne paket podatci će ostati sigurni. U svakom paketu mora biti barem jedan okvir koji služe za komunikaciju. Za analizu paketa koristi se Wireshark s kojim se mogu pogledati detalji o paketima. Pomoću Java programskog jezika možemo razviti QUIC protokol. Java je objektno-orijentirani programski jezik čiji se programi mogu pokrenuti na bilo kojem uređaju koje posjeduje JVM. Pri razvoju QUIC-a bitna je manipulacija bitovima jer pomoću njih se dolazi do određene informacije unutar poruke. Manipulacijom bitova upravljamo varijablama na razini bitova, a ne na razini njihovih stvarnih vrijednosti. Pomoću priključka Maven lako se može instalirati testNG koji koristimo za pisanje testova koji su nam potrebani za razvoj vođen testovima. Razvoj vođen testovima je način razvoja programa u kojem se pišu unit testovi prije programa. Cilj razvoja vođenog testovima je pisati čitljiv kod i kod koji se lako može testirati.

Ključne riječi

QUIC, paket, komunikacija, Wireshark, Java, razvoj vođen testovima, manipulacija bitova

Development of QUIC protocol

Summary

QUIC is a protocol based on the UDP protocol that has features of the TCP protocol. In communication between the server and the user, it uses various packages depending on the message. The packages are protected by encryption so that if someone intercepts the package, the data will remain secure. In each package, there must be at least one frame used for communication. Wireshark is used to analyze the packets, which can be used to view the details of the packets. Using the Java programming language, we can develop the QUIC protocol. Java is an object-oriented programming language whose programs can be run on any device that has a JVM. When developing QUIC, the manipulation of bits is important because it should be possible to get to certain information within the message. By bit manipulation, we manage variables by changing the bits, not their actual values. Using the Maven plugin, it is easy to install testNG, which we use to write the tests we need for test-driven development. Test-driven development is a method of program development in which unit tests are written before the program. The goal of test-driven development is to write readable and easily testable code.

Keywords

QUIC,package, communication, Wireshark, Java, test driven development, bit manipulation

1 | Uvod

Ovaj rad sadrži opis QUIC transportnog protokola te sadrži opis stvari koje su potrebne pri razvijanju vlastitog QUIC protokola u programskom jeziku JAVA pisanom u IntelliJ. QUIC je mješavina dva standardizirana protokola u cilju poboljšanja brzine i sigurnosti prijenosa podataka te se još uvijek radi na njegovom razvitku.

U poglavlju 2 dan je opis QUIC protokola i koje su mu prednosti u odnosu na druge protokole. Poglavlje 3 donosi opis procesa analiziranja paketa pomoću programa Wireshark. Opis Java programskog jezika i kako se koristi u razvoju QUIC-a dan je u poglavlju 4, zajedno sa primjerom. U poglavlju 5 dan je opis i primjer razvoja vođenog testovima, dok je u poglavlju 6 kroz primjere opisano kako manipulirati bitovima u programiranju.

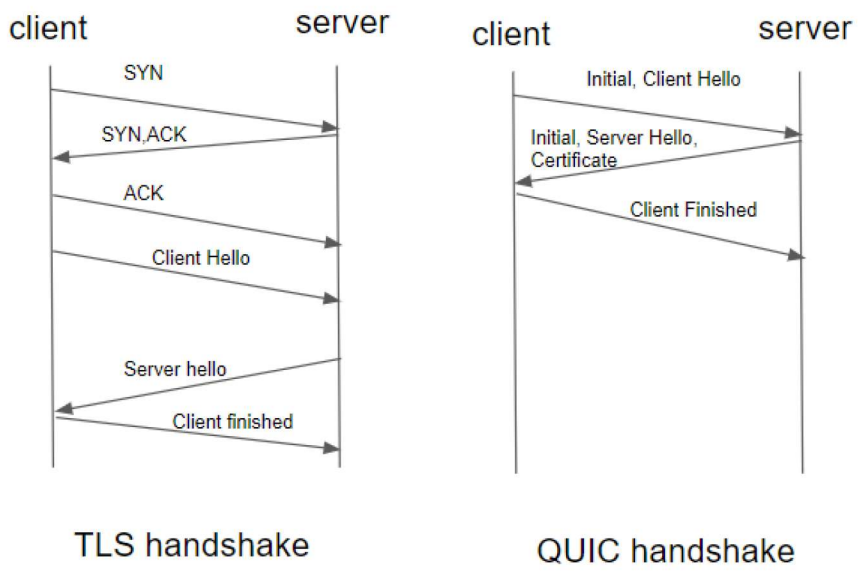
2 | QUIC

2.1 Povijest i korisnost

QUIC je transportni protokol koji je nastao 2012. godine u Google-u. Kad je tek nastao bilo je predloženo ime Brzi UDP transportni protokol (engl. Quick UDP transport protocol, QUIC), ali je protokol priznat pod nazivom QUIC od strane radnog tijela za razvoj interneta u dokumentu pod imenom RFC 9000 te se na ime QUIC ne gleda kao na akronim. QUIC služi kao alternativa TCP protokolu te to radi tako da stvori signal između dvaju uređaja pomoću UDP protokola koji oponaša rad TCP protokola. TCP kao protokol je teško zamijeniti jer je nastao još 1973. godine te je postao standard kod transportnih protokola jer nudi sigurnost tijekom prijenosa podataka. Protokoli se ne mjenjaju često jer je ispravnost rada najvažnija.

2.2 Potreba za QUIC

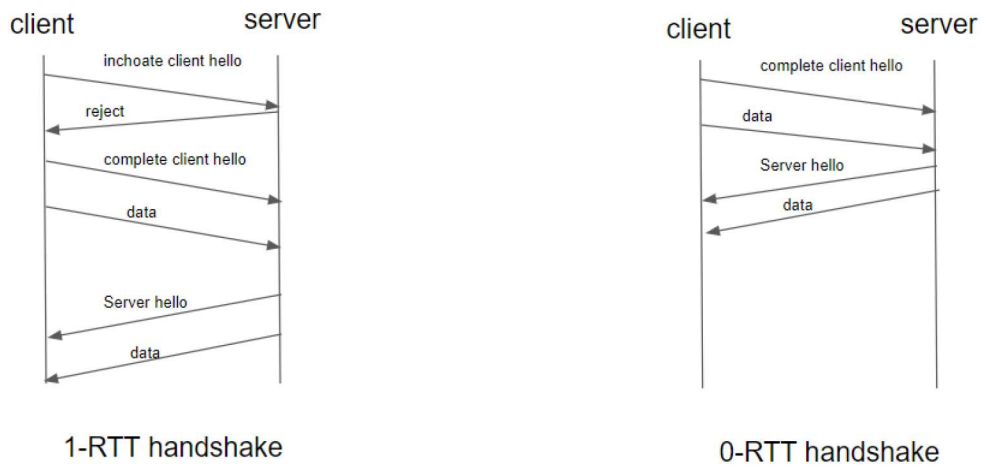
Razlika između UDP protokola i TCP protokola je ta što kod TCP protokola poslužitelj (engl. server) pri slanju paketa traži potvrdu od primatelja da je paket stigao te tako garantira da će klijent primiti pakete u točno određenom poretku, dok kod UDP protokola poslužitelj (engl. server) šalje pakete bez da traži potvrdu od korisnika (engl. client) što ga čini puno bržim, ali nesigurnijim. Tako je nastala potreba da se napravi protokol koji će imati značajke TCP-a, ali će se implementirati preko UDP-a kako bi dobili protokol koji je brži i sigurniji. Zbog tog razlog QUIC zovu i TCP/2. Koristi se na više od pola poveznica (engl. connections) na najpoznatijim tražilicama. Godine 2021. objavljen je dokument RFC-9000 kojim je QUIC postao standardiziran protokol. Pošto je QUIC relativno novi protokol još uvijek ima svojih mana koje se trebaju ispraviti. Određene sigurnosne stijene (engl. firewall) mogu stvarati probleme QUIC protokolu jer ne primaju UDP pakete koji se smatraju manje sigurnima od TCP paketa. Na određenim operativnim sustavima QUIC neće biti dobro optimiziran te se neće puno razlikovati od TCP-a po brzini.



Slika 1 *: Dijagram prikazuje razliku između QUIC i TLS protokola za uspostavu veze između poslužitelja (engl. server) i korisnika (engl. client)*

2.3 Vrste paketa

Kako bi povećao svoju efikasnost, QUIC ima više vrsta paketa te razlikujemo pakete s dugim i kratkim zaglavljem (engl. header). Paket s kraćim zaglavljem se dodatno još naziva 1-RTT paket te se koristi kad je veza već uspostavljena. Paketi s dugačkim zaglavljem (engl. Long header packets) su paketi koji se šalju prije nego što se ostvari 1-RTT veza. S tim paketima se ostvaruje veza između poslužitelja i korisnika te se određuje verzija QUIC-a. Verzija QUIC-a se određuje slanjem paketa za pregovaranje verzije QUIC-a. Tako poslužitelj nakon primljenog prvog paketa od klijenta šalje paket za pregovaranje verzije ako ne prepoznaje verziju QUIC-a kojom je klijent poslao prvi paket te bi tako korisnik u prvom paketu kojeg šalje trebao navesti sve verzije QUIC-a koje podržava. QUIC verzija definira se pomoću 32-bitnog nenegativnog cijelog broja (engl. unsigned int). Također se određuje ID svake konekcije zbog kojega će se unatoč promjeni internetske veze slanje moći nastaviti jer transport ne ovisi o IP-u veze nego isključivo o ID veze koji je poseban za svaku vezu između poslužitelja i korisnika. Postoji tip paketa 0-RTT koji može početi sa slanjem paketa s poslužitelja na klijent poslije prvog javljanja klijenta poslužitelju.



***Slika 2**: Dijagram prikazuje razliku između 1-RTT i 0-RTT QUIC protokola za uspostavu veze između poslužitelja i korisnika*

2.4 Zaštita, okviri i greške

QUIC je zaštićen enkripcijom tako da je slanje paketa sigurno te ako netko presretne paket neće moći ništa saznati iz njega bez ključa za dekriptiranje kojega ima samo korisnik. QUIC omogućuje u jednom usmjeravanju (engl. stream), slati pakete jednosmjerno i dvosmjerno te tijekom iste konekcije podržava više različitih usmjeravanja. Paket unutar usmjerenja mora sadržavati barem jedan okvir (engl. frame), a može sadržavati i više okvira. Okviri služe za komunikaciju poslužitelja i korisnika. Postoji više vrsta okvira ovisno o poruci koju si krajnje točke (engl. endpoint) žele poslati. To su:

- popunjavajući okvir (engl. padding frame) - koristi se za povećanje veličine paketa. Koristi se u početnom (engl. initial) paketu da bi zadovoljio minimalnu veličinu paketa,
- ping okvir - koristi se za provjeru konekcije,
- ACK okvir - koristi se za potvrđivanje primljenog paketa,
- okviri za zaustavljanje ili pokretanje novog usmjeravanja - zaustavi slanje (engl. stop sending), resetiraj usmjeravanje (engl. reset stream),
- okviri koji opisuju stanje usmjeravanja (engl. max stream data, max streams, stream data blocked),
- postoje okviri koji definiraju novi ID veze te okviri za terminiranje trenutne veze (novi ID veze (engl. new connection ID), povući ID veze (engl. retire connection ID), veza zatvorena (engl. connection close)).

Postoje kodovi za greške zapisani u 62-bitni nenegativni cijeli broj. Kodovi za greške (engl. error) dijele se na kod transport greške koji opisuju probleme u vezi te na kod protokolne greške koji se odnose na usmjeravanje te ovisno o problemu šaljemo određene okvire.

3 | Analiza paketa

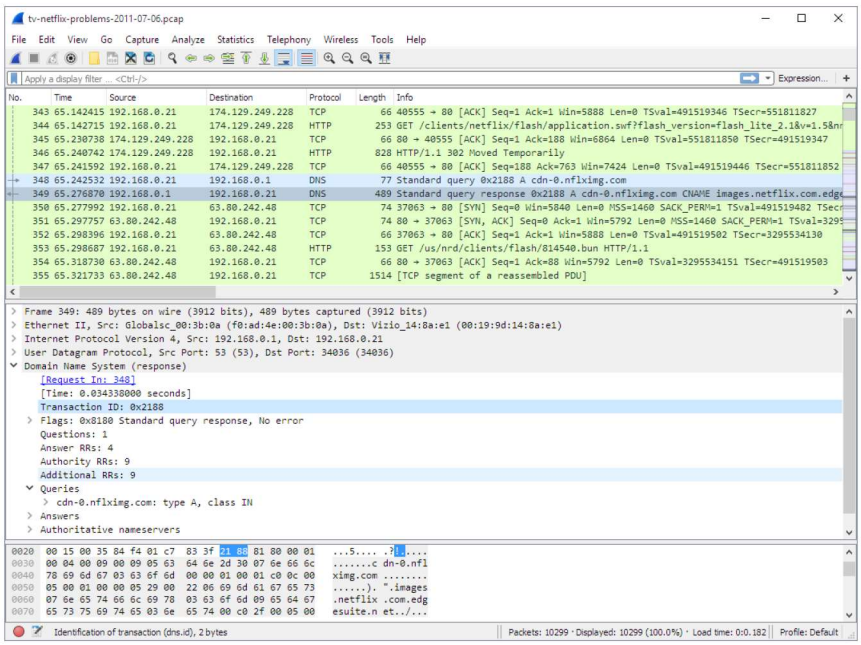
Za analizu paketa koristi se besplatni program Wireshark. Wireshark se koristi za mrežno rješavanje problema, za razvijanje softvera i transportnih protokola. Postoji verzija za razne operacijske sustave kao što su Linux, MacOS i Microsoft Windows. Wireshark je koristan jer ima dobro grafičko sučelje u kojem možemo vidjeti svaki paket te svakom od paketa možemo pristupiti te ih proučavati. Pomoću Wiresharka možemo vidjeti sav mrežni promet jednog uređaja te uživo možemo pratiti koje pakete uređaj prima i šalje. Možemo zadati pravila za koje će paket bojati u određenu boje te tako lako možemo pratiti pakete koji nas zanimaju. QUIC primjer kako radi Wireshark:

1. Otvorimo neku stranicu (npr. "www.youtube.com") koja stvara neki mrežni promet te primljene pakete i kojim protokolom su poslani možemo vidjeti na Wiresharku.

2. Na browseru izvezemo sigurnosni ključ za QUIC te ga dodamo Wiresharku koji će nam omogućiti čitanje primljenih paketa.

3. Filtriramo pakete po QUIC paketima.

Tako ćemo na Wiresharku dobiti listu QUIC paketa koji su poslani sa stranice koju smo otvorili te ćemo moći proučavati svaki dobiveni paket.



***Slika 3**: Slika Wireshark sučelja*

4 | Razvoj QUIC-a u Java programskom jeziku

4.1 Java u inteliJ

Java je objektno-orijentirani jezik baziran na klasama koji je jedan od najkorištenijih programskih jezika na svijetu te ga prema GitHubu koristi 9 milijuna programera. Razvio ga je inženjer James Gosling u tvrtki Sun Microsystem. Razvoj je počeo 1991. godine, a objavljen je 1995. godine. Velika prednost Jave u tom vremenu je što se program pisan u Javi mogao izvoditi bez preinaka na svim operacijskim sustavima koji posjeduju Java Virtual Machine (JVM). Izlazak Jave je potaknuo Microsoft na razvijanje C# i .NET platforme za alternativu otvorenog koda (engl. open source). Najnovija verzija je Java 21 koja je objavljena u rujnu 2023. godine.

IntelliJ IDEA je okružje za razvoj programa u jezicima koji su bazirani na JVM-u. Napisan je u Javi i dostupan je u javnoj i komercijalnoj verziji. IntelliJ pruža pomoć u kodiranju kao što su automatsko nadopunjavanje naredbi, skakanje brzo na određenu klasu ili deklaraciju, debugging i refaktoriranje koda. IntelliJ ima ugrađene alate i priključke (engl. plugin) koji nam pomažu pri razvoju programa.

Jedan od njih je Maven kojem je primarni cilj olakšavanje procesa građenja (engl. build), uniformno građenje sustava, davanje informacija o projektu te poticanje bolje prakse za razvoj programa. Tako nam Maven omogućava jednostavno instaliranje testNG. TestNG je okvir za testiranje (engl. testing framework) koji je inspiriran JUnitom koji je jači od njega te lakši za korištenje i podržava testiranje vođeno podatcima (engl. data-driven testing).

U javi pokušaj-uhvati (engl. try-catch) izrazima rješavamo iznimke u kodu. U pokušaj bloku napišemo kod kod kojega se može dogoditi iznimka te u uhvati blok pišemo kod koji će se izvršiti kada se dogodi greška u pokušaj bloku. Tako znamo koji dio programa je problematičan ako se dogodi greška. Sa slike 4 može se vidjeti primjer kako će se probati izvršiti funkcija copyOfRange s danim parametrima te ako se ona ne uspije izvršiti dogoditi će se greška.

```
try {
    a = Arrays.copyOfRange(data, start, stop);
} catch (ArrayIndexOutOfBoundsException e) {
    throw new BufferReadError(e.getMessage());
}
return a;
```

***Slika 4**: Primjer pokušaj-uhvati bloka*

```
public byte[] getData() {
    byte[] a = new byte [pos];
    for(int i=0;i<pos;i++){
        a[i]=data[i];
    }
    return a;
}
```

***Slika 5**: Primjer public funkcije.*

```
private final byte[] data;
9 usages
private final int capacity;
34 usages
private int pos;
```

***Slika 6**: Primjer privatnih varijabli.*

```
public enum QuicProtocolVersion {
    1 usage
    NEGOTIATION( value: 0),
    3 usages
    VERSION_1( value: 0x00000001),
    1 usage
    DRAFT_29( value: 0xFF00001D),
    1 usage
    DRAFT_30( value: 0xFF00001E),
    1 usage
    DRAFT_31( value: 0xFF00001F),
    1 usage
    DRAFT_32( value: 0xFF000020);
    public final byte value;
    6 usages  Vrkljan1
    QuicProtocolVersion(int value) { this.value = (byte) value; }

}
```

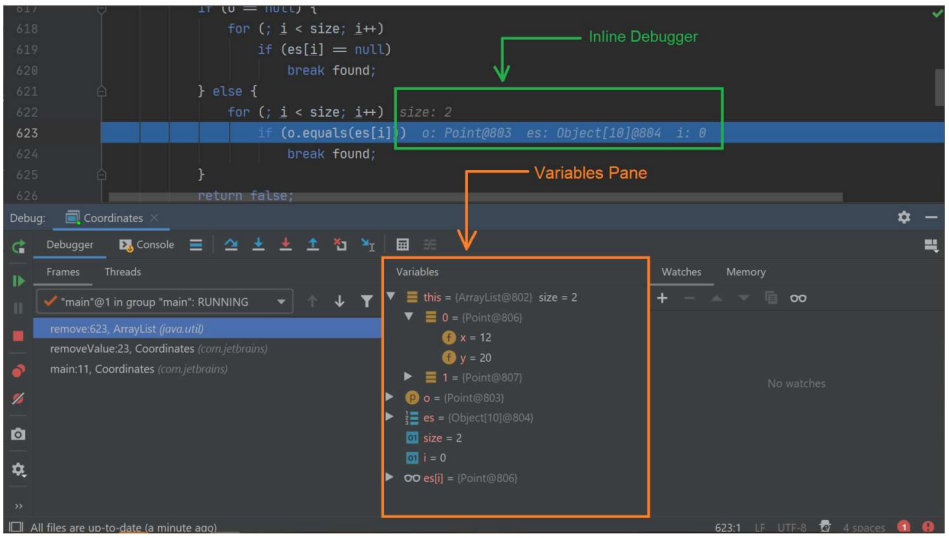
***Slika 7**: Primjer enum klase.*

4.2 Pronalaženje pogrešaka u IntelliJ

U razvijanju programa pronalaženje pogrešaka (engl. debugging) se definira kao proces traženja i rješavanja grešaka. Za traženje grešaka koristimo zaustavne točke (engl. breakpoint) koje definiraju u kojem dijelu programa će se program zaustaviti. U tom trenutku možemo vidjeti stanje svih varijabli te možemo pustiti program do iduće zaustavne točke te također u svakom trenutku možemo pogledati stanje varijabli. To je korisno jer možemo vrlo lako vidjeti ako nam se varijable ne mijenjaju kako je zamišljeno te se tako može uočiti u kojem dijelu programa je problem. Postoji niz funkcija za puštanje daljnjeg izvršavanja koda kao što su:

- prekorači (engl. step over) - izvrši se iduća linija koda,
- zakorači (engl. step into) - program za pronalaženje pogrešaka uđe u određenu metodu koju koristimo u metodi u kojoj se rješavamo grešaka,
- iskorači (engl. step out) - program za pronalaženje pogrešaka izađe iz određene metodu u metodu na kojoj smo pozvali program za pronalaženje pogrešaka,
- idi do pokazivača (engl. run to cursor) - izvrši se dio metode do pokazivača,
- ocijeni izraz (engl. evaluate expression) - daje nam mogućnost da izračunamo stanje izraza u nekom trenutku,

- nastavi program (engl. resume program) - nastavit će izvršavanje metode do iduće zaustavne točke ili do kraja metode,
- zaustavi (engl. stop) - izlazak iz program za pronalaženje pogrešaka.



***Slika 8**: Slika konzole programa za pronalaženje pogrešaka koja prikazuje stanje varijabli u 623. liniji koda*

5 | Razvoj vođen testovima

5.1 Način na koji se koristi

Razvoj vođen testovima (engl. test driven development, TDD) je vrsta razvoja softvera za koju se prvo pišu unit testovi te se potom piše program koji zadovoljava te iste testove. Slijed radnji koje radimo kod TDD-a:

1. Dodati novi test koji još neće imati svoju implementaciju.
2. Pokrenuti sve testove. Novi test mora pasti pošto još nemamo implementaciju za isti.
3. Napisati kod kojim će novi test proći.
4. Sada trebaju svi testovi proći.
5. Refaktoriranje koda kako bi kod bio kvalitetniji te radio na općenitom primjeru.

To ponavljamo za svaku funkcionalnost koju želimo dodati svome programu.

Unit testove pišemo pomoću potvrdnih (engl. assert) funkcija koje provjeravaju radi li metoda kako smo pretpostavili da će raditi. Postoji više potvrdnih funkcija kao što su:

- assertEquals - provjerava je li vrijednost metode za neki primjer jednaka očekivanom rezultatu,
- expectThrows - definiramo slučaj za koji nam metoda neće raditi te definiramo poruku koju ćemo dobiti ako se dogodi throw,
- assertTrue - provjerava je li vrijednost koju joj proslijedimo true,
- assertFalse - provjerava je li vrijednost koju joj proslijedimo false.

TDD-u nije glavna svrha istestirati sav kod koji napišemo nego dizajnirati kod tako da je čitljiv i da ga je lako testirati. Ako napravimo više jednostavnih testova za koje implementiramo funkcionalnost funkcije naš kod bit će pregledan te će se iz testova lako vidjeti što bi trebala biti funkcionalnost funkcije. Tako bi programer koji koristi TDD kao razvojnu metodu softvera trebao ukloniti jednostavne i lako ispravljive pogreške iz svoga koda te tako TDD svakoga programera čini

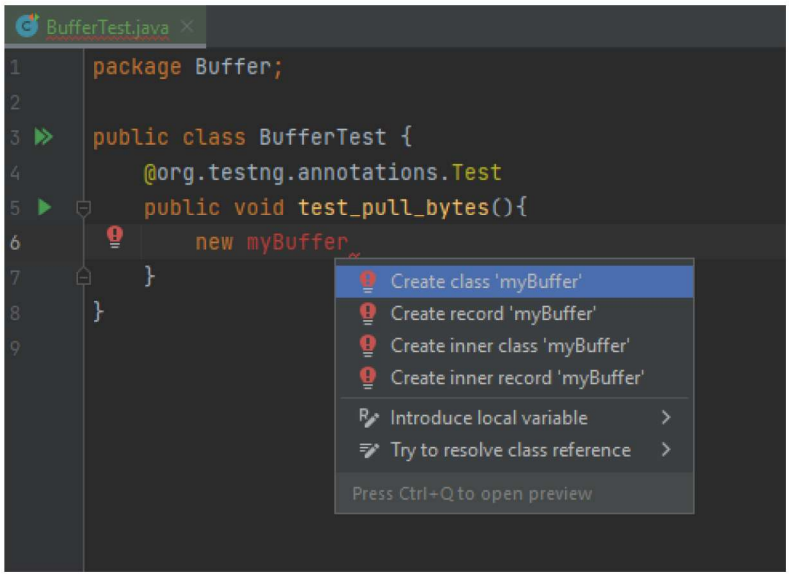
malo boljim nego što on je. Na temelju TDD-a razvijao se i BDD (engl. behavior driven development) kod kojega se ljudskim jezikom opisuje funkcionalnost pa se tek onda ide na razvoj te funkcionalnosti. BDD je primjenjiv za razvoj specifičnih softvera na temu na koju programeri nisu dovoljno stručni te im treba pomoć stručnjaka.

5.2 Primjer

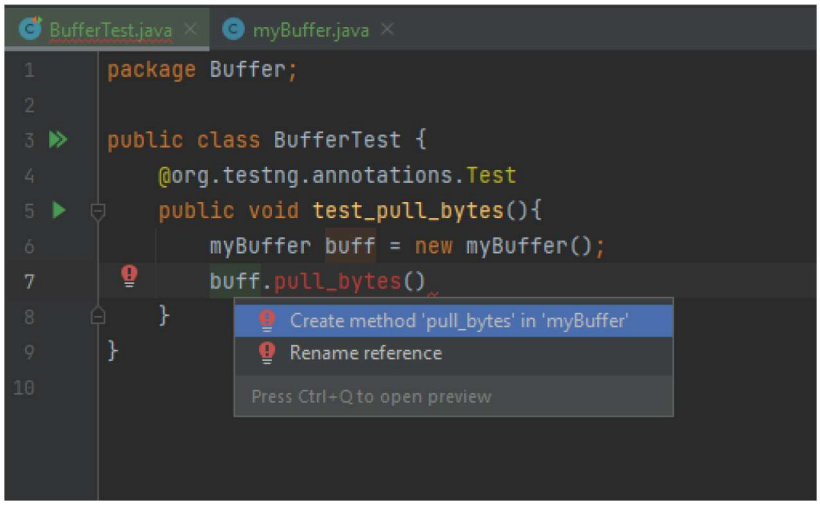
U ovom poglavlju, na temelju koraka iz prošlog poglavlja TDD će biti objašnjen na primjeru iz projekta razvoja funkcije pull_bytes pomoću koraka TDD-a.

1. Napisati testno tijelo (engl. test body)

   (a) definirati test i napraviti praznu klasu



   (b) napraviti varijablu te pozvati metodu koju želimo definirati i napraviti je u klasi



(c) zatim dodati potvrdni izraz (engl. assertion statment)

```
@org.testng.annotations.Test
public void test_pull_bytes() {
    byte[] data = Util.stringToByte("0x08 0x07 0x06 0x05 0x04 0x03 0x02 0x01");
    Buffer buf = new Buffer(data);
    byte[] a1 = Util.stringToByte("0x08 0x07 0x06");
    try {
        Assert.assertEquals(buf.pull_bytes( len: 3), a1);
    } catch (BufferReadError e) {
        throw new RuntimeException(e);
    }

    Assert.expectThrows(IllegalArgumentException.class, () -> buf.pull_bytes( len: -1));
}
```

2. pokrenuti test koji će pasti jer nema implementacije

3. napisati implementaciju

```
64      public byte[] pull_bytes(int len)throws BufferReadError{
65          byte [] a = Arrays.copyOfRange(data,pos, to: len+pos);
66          pos =pos + len;
67          if(len<0){
68              throw new BufferReadError("negative len");
69          }
70          if(capacity == 0 && len>0){
71              throw new BufferReadError("empty data");
72          }
73
74          return a;
75      }
```

4. pokrenuti test koji bi trebao proći

```
================================================
Default Suite
Total tests run: 1, Passes: 1, Failures: 0, Skips: 0
================================================


Process finished with exit code 0
```

6 | Manipulacija bitovima

Manipuliranje bitovima radimo bitwise operacijama kojima varijablama mijenjamo vrijednost na razini bitova, a ne na razini njihovih stvarnih vrijednosti. U to spadaju bitwise operatori, pomicanje bitova (engl. bit shifts) te mijenjanje baznog sustava u kojem je broj zapisan. Bitwise operatori su: NOT, AND, OR, XOR i matematičke jednakosti.

Aritmetičko pomicanje bitova pomiče bitove u lijevo ili desno ovisno radi li se o lijevom ili desnom pomicanju bitova. Bitovi koji se pomaknu van se ignoriraju. Lijevi pomak nad integerom ga množi s 2^n te desni pomak ga dijeli s 2^n gdje je n broj pomaka te to u programskom kodu zapisujemo kao

```
x= y<<2,
```

što znači da je x jednak y kod kojega je bit pomaknut za dva mjesta. Primjer bitwise operacije može se vidjeti na slici 10 koja prikazuje kako pomičemo bitove u lijevo ovisno o veličini prvog bita. Ako podatak nije zapisan kao bajt treba ga transformirati u bajt kao što je prikazano na slici 11.

```
public long pull_uint_var() throws BufferReadError{
    long value;
    switch(Math.abs((data[pos] & 0xFF)>> 6)){
        case 0:
            value = data[pos++] & 0x3F;
            break;
        case 1:
            value = ((data[pos] & 0x3F) << 8 |
                    data[pos+1] & 0xFF);
            pos +=2;
            break;
        case 2:
            value = (data[pos] & 0x3F) << 24 |
                    (data[pos + 1] & 0xFF)  << 16 |
                    (data[pos + 2] & 0xFF) << 8 |
                    (data[pos + 3] & 0xFF);
            pos += 4;
            break;
        default:
            long a = (long) (data[pos + 4] & 0xFF) << 24L;
            value = (long) (data[pos] & 0x3F) << 56 |
                    (long) (data[pos + 1] & 0xFF) << 48 |
                    (long) (data[pos + 2] & 0xFF) << 40 |
                    (long) (data[pos + 3] & 0xFF) << 32 |
                    (long) (data[pos + 4] & 0xFF) << 24 |
                    (data[pos+ 5] & 0xFF)  << 16 |
                    (data[pos+ 6] & 0xFF)  << 8 |
                    (data[pos+ 7] & 0xFF) ;
            pos += 8;
            break;

    }
    return value;

}
```

***Slika 10**: Primjer primjena bitwise operacija u QUIC protokolu.*

```java
public static byte[] intToByte(long value){
    if(value==0){
        return new byte[1];
    }
    byte[] a = new byte[16];
    int i = 0;
    while(value>0){
        a[i]= (byte)(value%16);
        value = value/16;
        i++;
    }
    int j =0;
    int k = i;
    i--;
    byte []rev = new byte[i+1];
    while(j<k){
        rev[i] = a[j];
        j++;
        i--;
    }
    byte[]ret = new byte[(j+1)/2];
    long temp=0;
    for(i=0;i<rev.length;i++){

        if(i%2==0){
            temp = (int)rev[i]*16;
        }else{
            ret[i/2] = (byte)((temp+(int)rev[i]) & 0xFF);
            temp=0;
        }
    }
    if (i%2==1) {
        ret[i/2] = (byte) ((temp/16) & 0xFF);
        return ret;
    }

    return ret ;
}
```

***Slika 11**: Primjer funkcije koja pretvara vrijednost iz broja s bazom 10 u broj s bazom 16.*

Literatura

[1] A. Binstock , Java's 20 Years of Innovation, Forbes, 20. svibnja 2015. URL: https://www.forbes.com/sites/oracle/2015/05/20/javas-20-years-of-innovation/?sh=62bf64b411d7

[2] J. Iyengar, Ed. , M. Thomson, Ed., *RFC 9000 ,QUIC: A UDP-Based Multiplexed and Secure Transport, svibanj 2021.*

[3] *Web izvor dostupan na:* https://maven.apache.org/what-is-maven.html.

[4] *Web izvor dostupan na:* https://www.guru99.com/test-driven-development.html.

[5] *Web izvor dostupan na:* https://code.tutsplus.com/articles/understanding-bitwise-operators–active-11301.

[6] *Web izvor dostupan na:* https://www.varonis.com/blog/how-to-use-wireshark.

[7] *Web izvor dostupan na:* https://www.jetbrains.com/help/idea/tdd-with-intellij-idea.html\#summary

[8] *Web izvor dostupan na:* http://www.agilenutshell.com/test_driven_development