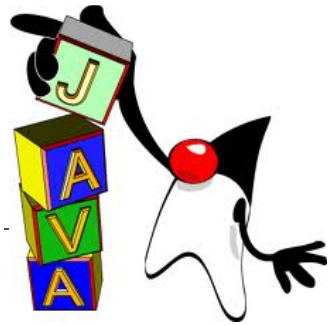
Strukturiranje kode

Programiranje 2, Tomaž Dobravec

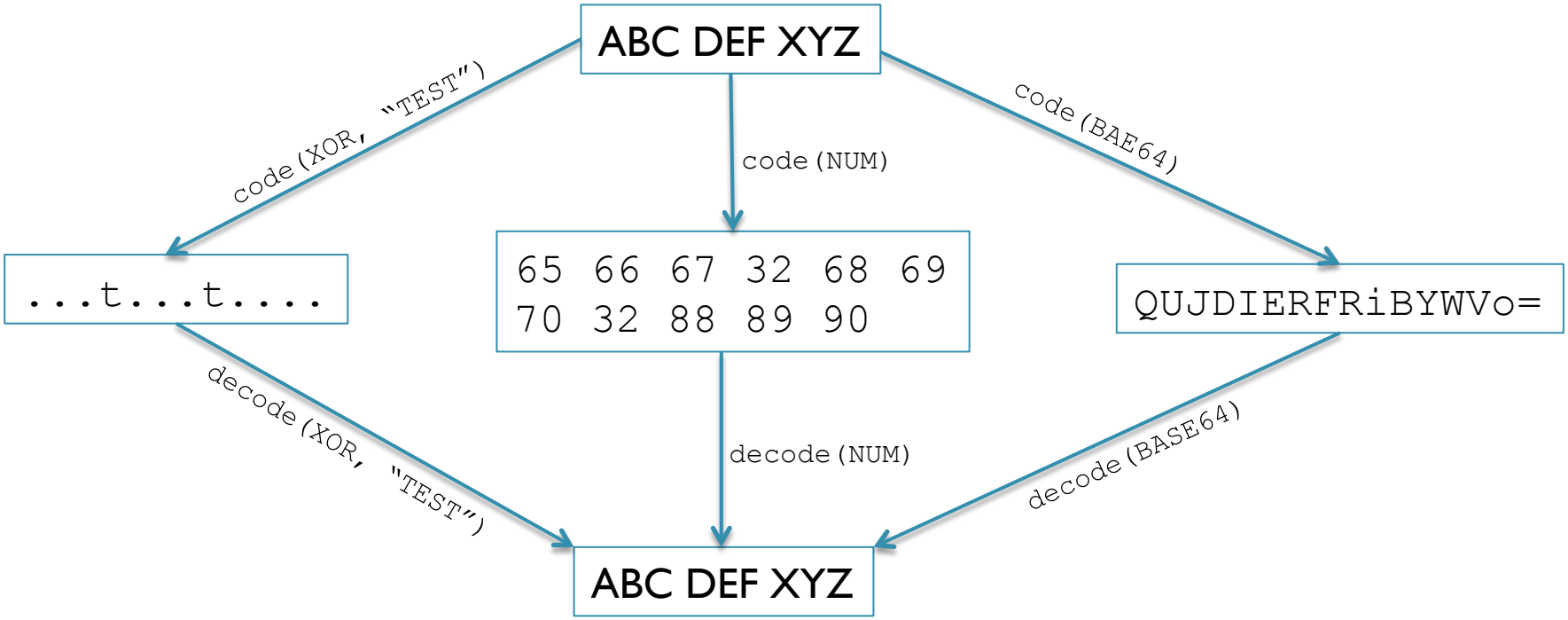
Naloga

Kodiranje datotek 1/5



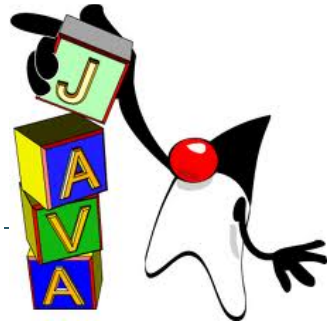
si.fri.kodiranje.*

Naloga: napiši program za kodiranje in dekodiranje datotek, ki uporablja tri algoritme kodiranja: XOR, NUM in BASE64. Pri klicu programa poveš smer delovanja (code / decode) ter ime vhodne in izhodne datoteke. V primeru kodiranja tipa XOR, je podan tudi ključ.



Naloga

Kodiranje datotek 2/5

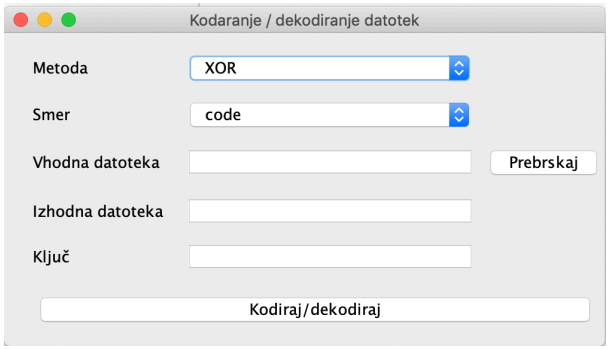


si.fri.kodiranje.*

- Program naj omogoča poganjanje iz ukazne vrstice:

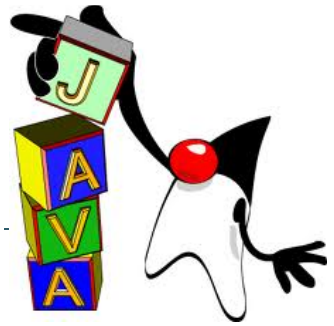
```
Default                                                    ⌥⌘1
kepica:code_jar tomaz$ java -jar code.jar XOR code a.txt a.out abc
kepica:code_jar tomaz$ ls
a.out           a.txt           a2.txt          code.jar
kepica:code_jar tomaz$ java -jar code.jar XOR decode a.out a2.txt abc
kepica:code_jar tomaz$ diff a.txt a2.txt
kepica:code_jar tomaz$
```

- Če argumenti niso podani, naj program izriše okno:



Naloga

Kodiranje datotek



Štirje razredi

Trije paketi

si.fri.kodiranje.main

Kodiranje.java

```
void main(args)
void doKodiranje(nacin, smer, vhod, izhod, kljuc)
```

Kodirniki.java

```
String kodirajNUM   (vsebina)
String kodirajXOR   (vsebina, kljuc)
String kodirajB64   (vsebina)
String odkodirajNUM(vsebina)
String odkodirajXOR(vsebina, kljuc)
String odkodirajB64(vsebina)
```

si.fri.kodiranje.io

BeriPisi.java

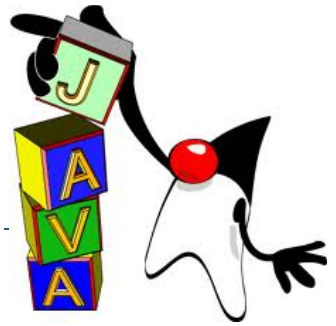
```
String preberiDatoteko  (imeDatoteke)
void   zapisiDatoteko   (imeDatoteke, vsebina)
```

si.fri.kodiranje.gui

KodiranjeOkno.java

```
void prikaziOkno()
void kodirajDekodiraj()
```

Kodiranje datotek



Logika delovanja programa (1/2)

Glavni program: metoda `Kodiranje.main()`:

- če argumenti obstajajo: prebere argumente in kliče metodo `Kodiranje.doKodiranje(nacin, smer, vhod, izhod, kljuc)`,
- če argumenti ne obstajajo: kliče `KodiranjeOkno.prikaziOkno()`.

Razred `KodiranjeOkno`:

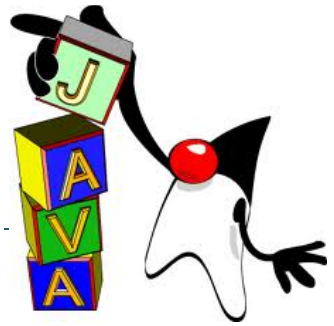
- metoda `prikaziOkno()`: prikaže okno, kamor uporabnik lahko vpiše parametre,
- ob kliku na gumb `"Kodiraj/Dekodiraj"` se kliče metoda `kodirajOdkodiraj()`
- metoda `kodirajOdkodiraj()`: prebere parametre iz vpisnih polj in kliče `Kodiranje.odKodiranje(nacin, smer, vhod, izhod, kljuc)`

Razred `BeriPisi`

- metoda `preberiDatoteko(imeDatoteke)`: prebere datoteko in vrne njeno vsebino kot niz,
- metoda `zapisiDatoteko(imeDatoteke, vsebina)`: v datoteko zapiše podano vsebino.

Razred `Kodirniki`

- metode tega razreda prejmejo vsebino in jo zakodirajo oziroma odkorirajo, vsaka po svojem postopku



Logika delovanja programa (2/2)

Metoda `Kodiranje.doKodiranje(nacin, smer, vhod, izhod, kljuc)`:

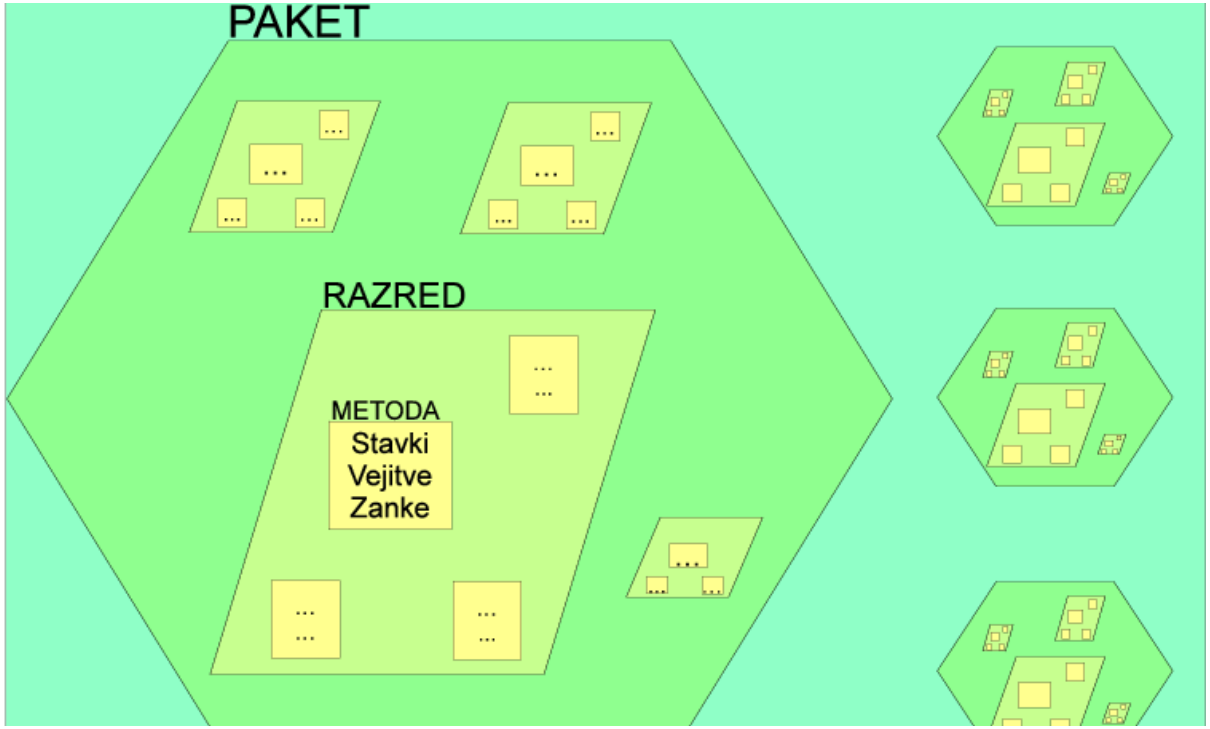
- prebere vsebino datoteke `vhod`
- glede na vrednost parametrov `nacin` in `smer` kliče eno od metod razreda `Kodirniki` in s tem zakodira/odkodira prebrano vsebino datoteke
- spremenjeno vsebino datoteke zapiše v `izhod`

Strukturiranje kode

Javanski program:

- stavki
- vejitve
- zanke
- metode
- razredi
- paketi

PROGRAM



Zanke `do`, `while` in `for`

- Java pozna tri zanke: `do`, `while`, `for`

Primer: tri zanke, ki naredijo isto:

```
int i=0;          | int i=0;          | int i;
do {              | while (i<=10) {   | for (i=1;i<=10;i++) {
  i=i+1;          |   i=i+1;          |   ....
  ....            |   ....            | }
} while (i<10)    | }                 |
```

do **while** **for**

Zanke `do`, `while` in `for`

Kdaj uporabim katero zanko?

- Če je izvajanje zanke nadzorovano s pomočjo nekega števca, običajno uporabimo `for`, če je pogoj zanke drugačne oblike, pa `do` ali `while` zanko.

- Primer smiselne uporabe zanke `while`:

```
boolean nasel = false;
while (!nasel) {
   … // v telesu zanke se spremeni
   … // vrednost spr. nasel
}
```

Vejitveni stavek `if-else`

- Vejitveni stavek `if-else` ima dve obliki:
  - `if` stavek brez dodanega `else` dela - primer `(a)`
  - `if` stavek v kombinaciji z `else` delom - primer `(b)`

- Java ne pozna stavka `elif`; to lahko nadomestimo s stavkom `else`, ki vsebuje nov `if-else` stavek - primer `(c)`

```
if (a==1)
  b=1;
```

(a)

```
if (x==5) {
  c=1;d=3;
} else {
  c=4;
}
```

(b)

```
if (q<5) {
  d=3;
} else if (q<2) {
  d=2;
} else {
  d=1;
}
```

(c)

Vejitveni stavek `switch`

- Zaporedje `if-else` stavkov, ki se vsi **nanašajo na isto spremenljivko**, lahko nadomestimo s stavkom `switch`.

- **Primer:** zaporedje `if-else` stavkov in stavek `switch` z istim učinkom

```
if (n==1)
  f=1;
else if (n==2)
  f=1;
else if (n==3)
  f=2;
else if (n==4)
  f=3;
else if (n==5)
  f=5;
```

```
switch (n) {
  case 1: f=1; break;
  case 2: f=1; break;
  case 3: f=2; break;
  case 4: f=3; break;
  case 5: f=5; break;
}
```

Pomembno: stavek `break` na koncu vsake veje.

Metode

- Program v javi je sestavljen iz metod.
- Preprosti programi imajo samo eno metodo (`main`), zahtevnejši jih imajo več.
- Metode predstavljajo ločene celote programa; s pomočjo metod program postane preglednejši.
- Metodo lahko kličem večkrat. Z združevanjem kode v metode se izognem ponavljanju istih delov programa. Če je treba ponavljajočo kodo popraviti, popravim na enem samem mestu.

Parametri metode in rezultat

- Metoda (lahko) prejme parametre in vrne rezultat

```
static double sredina(double x, double y) {
  return (x+y)/2
}
```

- Metodo *pokličemo, na primer,* takole:

```
double a = sredina(5, 7); // a = 6.0
```

Preoblaganje metod

V razredu imamo lahko več metod z **istim** imenom a z **različnim** številom in/ali tipom parametrov (method overloading)

```
public static void izpisi(int x, String napis) {
  // … metoda izpise x presledkov ter ”napis”
}
public static void izpisi(String napis) {
  // privzeto število presledkov: 0
  izpisi(0, napis);
}
public static void izpisi() {
  // privzeto besedilo izpisa je ”OPOZORILO!”
  izpisi(”OPOZORILO!”);
}
```

Preoblaganje delno nadomešča privzete vrednosti parametrov

Program v več razredih

- Krajši programi so v celoti napisani v enem razredu, koda zahtevnejših programov pa je razdeljena v več razredov.
- V posameznem razredu je zbrana koda, ki smiselno sestavlja celoto.
- Razredi lahko uporabljajo drug drugega (atributi in metode iz enega razreda so vidni tudi v drugem razredu – če seveda niso skriti)

Paketi

Na nivoju operacijskega sistema: paket == mapa (folder).

Namen in uporaba:

- smiselno združevanje razredov,
- preprečevanje konfliktov imen,
- lažja distribucija programov.

Obstoječi paketi

- Java s seboj prinaša mnogo paketov: `java.applet`, `java.awt`, `java.io`, `java.lang`, `...`
- Pakete lahko dobimo v `jar` datotekah, ki jih prenesemo s spleta.
- Pakete ustvarimo sami.

Imena paketov

Paket smiselno poimenujemo (glede na njegov namen in vsebino razredov).

- Primer: primerno ime za paket, ki vsebuje metode za delo z matematičnimi funkcijami, je, na primer, `matematika`.

Za preprečevanje konflikta imen paketu dodamo "predpono", ki enolično opisuje, kdo je avtor paketa. Običajno je predpona sestavljena in obrnjenega internetnega naslova.

Primer: podjetje *Kava d.o.o.* s spletno stranjo

```
http://kava.si
```

bo svojim paketom dodalo predpono `si.kava`. Paket z matematičnimi funkcijami se bo pri njih imenoval `si.kava.matematika`.

Imena paketov - primer

- Podjetje Apache ponuja zbirko uporabnih knjižnic (razredi, metode, …) za Javo s skupnim imenom Apache Commons
- Domača stran projekta je https://commons.apache.org/
- Vsi razredi tega projekta so v paketu

```
org.apache.commons
```

Primer: kopiranje datoteke s pomočjo Apache Commons (`FileUtils`).

Uporaba paketov

- Razred, ki ni v našem paketu, moramo pred uporabo uvoziti:

```
import java.util.Random;
Random r = new Random();
```

- Lahko uvozimo tudi celoten paket (in s tem vse razrede, ki so v njem):

```
import java.util.*
```

- Če ne želimo uvažati, lahko pri uporabi podamo celotno ime:

```
java.util.Random r = new java.util.Random();
```

Avtomatski uvoz paketa `java.lang`

Razred `Math` se nahaja v paketu `java.lang`.

Zakaj lahko napišemo

```
double pi = Math.PI;
```

brez uvoza `import java.lang.Math`?

Ker java paket `java.lang` avtomatsko uvozi sama!

Opomba: paket `java.lang` je edini paket z avtomatskim nalaganjem!

Statičen uvoz

Statične metode in atribute lahko uporabljamo brez navedbe razreda, če jih prej *statično uvozimo*.

Namesto

```
double pi = Math.PI;
```

lahko napišemo

```
import static java.lang.Math.*;
…
double pi = PI;
```

Uporaba statičnega uvoza ni priporočljiva!

Pot do paketov

```
package si.fri.math.geometrija;

public class Trikotnik {
  // stranice trikotnika
  static int a=3;
  static int b=4;
  static int c=5;

  public static void izpis() {
    System.out.printf("Obseg trikotnika s stranicami "
            + "%d, %d in %d je %d\n", a, b, c, a+b+c);
  }

  public static void main(String[] args) {
    izpis();
  }
}
```

```
D:\> cd program\si\fri\math\geometrija
D:\program\si\fri\math\geometrija> dir
Trikotnik.java
D:\program\si\fri\math\geometrija>javac Trikotnik.java
D:\program\si\fri\math\geometrija>java Trikotnik.java
Exception in thread "main" java.lang.NoClassDefFoundError ...
D:\program\si\fri\math\geometrija>cd \program
D:\program>java Trikotnik
Exception in thread "main" java.lang.NoClassDefFoundError ...
D:\program>java si.fri.math.geometrija.Trikotnik
Obseg trikotnika s stranicami 3, 4 in 5 je 12

D:\program>cd ..
D:\>java si.fri.math.geom.Trikotnik
Exception in thread "main" java.lang.NoClassDefFoundError ...
D:\>java program.si.fri.math.geom.Trikotnik
Exception in thread "main" java.lang.SecurityException ...
```

Pot do paketov


```
D:\>java -cp d:\program si.fri.math.geometrija.Trikotnik
Obseg trikotnika s stranicami 3, 4 in 5 je 12


D:\>SET CLASSPATH=d:\program
D:\>java si.fri.math.geom.Trikotnik
Obseg trikotnika s stranicami 3, 4 in 5 je 12
D:\>c:
C:\>cd Windows
C:\WINDOWS>java si.fri.math.geom.Trikotnik
Obseg trikotnika s stranicami 3, 4 in 5 je 12
```

Dostopnostna določila

Java pozna 4 dostopnostna določila (anlg. access modifiers):

Določilo	Pomen
`public`	dostopno povsod in vsakomur
`protected`	dostopno v vseh razredih tega paketa ter v vseh podrazredih (tudi, če niso v tem paketu)
`default` (brez določila)	dostopno v vseh razredih tega paketa
`private`	dostopno samo v tem razredu

Čitljivost javanske kode

- Zelo pomembno je, da je programska koda ČITLJIVA!
- K čitljivosti pripomorejo:
  - uporaba dogovorjenih pravil o poimenovanju spremenljivk metod, razredov in paketov,
  - uporaba dogovorjenih pravil o oblikovanju programa (zamiki) in
  - razumno komentiranje kode.

Podrobneje o čitljivosti kode:

http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

Komentarji v *javi*

Java pozna tri tipe komentarjev:

// komentar	enovrstični komentarji; komentar je vse od znaka // do konca vrstice
/* komenar */	večvrstični komentarji; komentar je vse, kar je ujeto med znaka /* in */
/** komantar */	komentar za dokumentacijo (angl. *doc comments*); tak komentar se pojavi tik pred deklaracijo identifikatorja (spremenljivke, metode, razreda, ...) in se uporablja za avtomatsko generiranje dokumentacije;

Dokumentacijski komentarji

Na podlagi dokumentacijskih komentarjev Java avtomatsko zgradi HTML dokumentacijo.

- Del dokumentacijskih komentarjev so tudi značke:

Značka	primer uporabe
@see	Povezave v rubriki "Glej tudi"
{@link}	Hiperpovezave med opisnim besedilom
@param	Opis parametrov metode
@return	Opis rezultata metode
@throws	Opis izjem, ki jih lahko vrze metoda
@author	Opis avtorja metode
@version	Stevilka verzije