Java programska kartica

Razred

`[public|final|abstract] class` *name*
{ *spremenljivke in metode razreda* }

Vstopna točka programa je statična metoda:
`public static void main(String[]` *args*`)`

Argumenti ob zagonu programa se preslikajo v polje nizov, ki je parameter metode `main`.

Izpeljani razred:
`class` *name* `extends` *basename* `[implements` *ifname*`]`

Vsi razredi so izpeljani iz osnovnega razreda `Object`, ki ima tudi metodo `String toString()`.

Spremenljivke in metode

enostavni tipi: `char, byte, short, int, long, float, double, boolean`
razredi enostavnih tipov: `Character, Byte, Short, Integer, Long, Float, Double, Boolean`

nekateri modifikatorji spremenljivk in metod:
`static, public, private, protected, final`
pomen modifikatorjev dostopa

- `public` – dostop omogočen od kjerkoli
- `protected` – dostop omogočen iz razreda in vseh njegovih podedovanih razredov
- `private` – dostop omogočen le znotraj razreda
- brez – dostop omogočen znotraj paketa, v katerem je deklaracija

Izjeme

Lovljenje izjem:

```
try {
  stavki, ki lahko sprožijo izjemo
}
catch (TipIzjeme1 ex) {
  stavki, ki opravijo z izjemo tipa TipIzjeme1, če do nje pride
}
catch (TipIzjeme2 ex) {
  stavki, ki opravijo z izjemo tipa TipIzjeme2, če do nje pride
}
finally {
  stavki, ki se vedno izvršijo, če je prišlo do izjeme ali ne
}
```

Nekateri pogosti tipi izjem:

- `ArithmeticException` – aritmetična napaka kot npr. deljenje z ničlo
- `ArrayIndexOutOfBoundsException` – napaka ob dostopu do elementa polja, ki ne obstaja (premajhen ali prevelik indeks polja)
- `NullPointerException` – neveljavna uporaba reference na `null`
- `NumberFormatException` – neveljavna pretvorba niza v število
- `IOException` – izjema pri delu z V/I tokovi (npr. pri branju iz datoteke)
- `Exception` – osnovni razred, iz katerega so podedovane ostale izjeme (`catch(Exception)` ujame vse tipe izjem)

Matematične funkcije `(java.lang.Math)`

Metode se nahajajo v razredu `Math` in so statične (kličemo jih kot `Math.ime_metode()`). Odvisno od metode so lahko argumenti in vrnjene vrednosti različnih tipov (`int`, `float`, `double`), vse funkcije podpirajo tip `double`.

- *trigonometrične*: `sin(x)`, `cos(x)`, `tan(x)`
- *inverzne* trig: `asin(x)`, `acos(x)`, `atan(x)`
- *logaritmi*: `log(x)` - naravni logaritem
- *potence*: `exp(x)` - e^x, `pow(x,y)` - x^y, `sqrt(x)` - kvadratni koren
- *zaokroževanje*: `ceil(x)` - zaokroži navzgor, `floor(x)` - zaokroži navzdol, `round(x)` - zaokroži
- *ostalo:* `abs(x)` - absolutna vrednost, `max(x,y)` - največja vrednost, `min(x,y)` - najmanjša vrednost, `random()` - naključna vrednost

Konverzije `(java.lang.*)`

Razredi enostavnih tipov nudijo metode za pretvorbo iz nizov v osnovne tipe in obratno. Primer za cela števila sta metodi:

- `static int Integer.parseInt(String` *s*`)` - pretvori niz v celo število oz. sproži `NumberFormatException`, če število ni v pravem formatu.
- `static String Integer.toString(int` *i*`)` - pretvori celo število v niz

Ekvivalentne metode veljajo za ostale tipe (npr. za `double` metodi `Double.parseDouble()`, `Double.toString()`).

Operacije z znaki `(java.lang.Character)`

Razred `Character` nudi več statičnih metod za delo z znaki, med njimi metode za preverjanje tipov znakov:

`boolean isDigit(char ch)`	je številka
`boolean isLetter(char ch)`	je črka
`boolean isLowerCase(char ch)`	je mala črka
`boolean isUpperCase(char ch)`	je velika črka
`boolean isWhiteSpace(char ch)`	je prazen prostor

Vse metode so statične in vrnejo vrednost true, če znak spada v opisano kategorijo znakov, in false, če tja ne spada.
Statični metodi za pretvorbo znakov sta:

`char toLowerCase(char ch)`	pretvori v malo črko
`char toUpperCase(char ch)`	pretvori v veliko črko

Delo nizi `(java.lang.String(Buffer) )`

Razreda `String` in `StringBuffer` sta namenjena delu z nizi. Objekt tipa `String` predstavlja nespremenljiv niz; niz, shranjen v razredu `StringBuffer`, pa lahko spreminjamo.

Nekatere metode razreda `String`:

- `char charAt(int` *ix*`)` – vrne znak na indexu *ix*
- `int compareTo(String` *str*`)` – primerja niz s *str*. Vrne 0, če sta niza enaka; <0, če je niz manjši od *str*; >0, če je niz večji od *str*. Metoda `compareToIgnoreCase()` naredi enako brez upoštevanja velikih/malih črk.
- `int length()` – vrne dolžino niza
- `int indexOf(int` *ch*`|String` *str*`)` – vrne mesto podanega znaka *ch* ali niza *str* oz. -1, če le-tega ne najde
- `String replace(char` *oc*`, char` *nc*`)` – zamenja vse pojavitve znaka *oc* z *nc* in vrne novi niz
- `String[] split(String` *rx*`)` - razbije niz glede na podan regularni izraz *rx* na polje nizov in to polje vrne (npr. če za izraz podamo presledek " ", niz razbije na besede in vrne polje besed).
- `String substring(int` *bi*`, int` *ei*`)` - vrne podniz niza, ki se začne na znaku *bi* in konča na *ei*-1.
- `String toLowerCase()` in `toUpperCase()` – metodi pretvorita niz v male oz. velike črke in vrneta novi niz
- `String trim()` - odstrani vse presledke iz začetka in konca niza in vrne novi niz
- operator `+` : operacija *str1* + *str2* doda niz *str2* na konec niza *str1* in vrne novi niz

Java programska kartica

Nekatere metode razreda `StringBuffer`:

- `StringBuffer append(char` *c*`|String` *str*`)` – doda znak *c* ali niz *str* na konec niza in vrne samega sebe
- `char charAt(int` *ix*`)` – vrne znak na indeksu *ix*
- `StringBuffer delete(int` *s*`, int` *e*`)` – zbriše znake od pozicije *s* do *e*-1 in vrne samega sebe
- `int indexOf(String` *str*`)` – vrne mesto začetka niza *str* v nizu ali -1, če niza *str* ne najde v nizu
- `StringBuffer insert(int` *pos*`, char` *c*`|String` *s*`)` – vrine znak *c* ali niz *s* na pozicijo *pos* v nizu
- `int length()` – vrne dolžino niza
- `StringBuffer replace(int` *s*`, int` *e*`, String` *str*`)` – zamenja znake med *s* in *e*-1 z nizom *str*. Vrne sebe.
- `String substring(int` *bi*`, int` *ei*`)` - vrne podniz niza, ki se začne na znaku *bi* in konča na *ei*-1.
- `String toString()` - pretvori `StringBuffer` v `String`

Delo z V/I tokovi (`java.io.*`)

Standardni vhod

Branje s standardnega vhoda omogoča statični objekt `in` (tipa `InputStream`) razreda `System`. Nudi osnovno metodo:

- `int read() throws IOException`
  Metoda prebere bajt s standardnega vhoda in vrne njegovo vrednost ali -1 ob koncu branja.

Standardni izhod

Pisanje na standardni izhod omogoča statični objekt `out` (tipa `PrintWriter`) razreda `System`. Metode razreda `PrintWriter` so opisane v nadaljevanju.

Delo z datotekami

Razred `File` lahko uporabljamo za delo z datotekami:

- `File(String` *name*`)` – konstruktor, ki naredi objekt, povezan z datoteko po imenu *name*
- `boolean createNewFile()` – naredi novo datoteko
- `boolean delete()` – zbriše datoteko in vrne `true`, če je izbris uspešen
- `boolean exists()` – vrne `true`, če datoteka obstaja
- `String getName()` – vrne ime datoteke
- `String getPath()` – vrne pot do datoteke
- `boolean isDirectory()` – vrne `true`, če je datoteka direktorij
- `String[] list()` – vrne imena datotek, ki so v direktoriju
- `boolean mkdir()` – naredi direktorij in vrne `true`, če ga je uspešno naredil
- `boolean renameTo(File` *dest*`)` – preimenuje datoteko v datoteko *dest* in vrne `true`, če je uspešen

Dostop do datoteke

Do datoteke lahko dostopamo preko razredov `FileReader` in `FileWriter`, ki predstavljata bralnik oz. pisalnik.
Konstruktor, ki naredi objekt za branje iz datoteke *fileName*:

- `FileReader(String` *fileName*`) throws FileNotFoundException`

Konstruktor, ki naredi objekt za pisanje v datoteko *fileName*:

- `FileWriter(String` *fileName*`) throws FileNotFoundException`

Branje preko medpomnilnika – BufferedReader

Iz poljubnega tekstovnega vhodnega toka (std. vhod ali datoteka) lahko beremo posamezne znake ali cele vrstice z metodami razreda `BufferedReader`:

- `BufferedReader(Reader` *in*`)` – konstruktor, ki naredi nov objekt na podlagi podanega bralnika. Ta je lahko npr. bralnik datoteke (`FileReader`) ali bralnik vhodnega toka (`InputStreamReader`)
- `int read() throws IOException` – prebere znak in vrne njegovo ASCII kodo ali -1 ob koncu vhodnega toka
- `String readLine() throws IOException` – prebere vrstico in jo vrne, ali pa vrne `null` ob koncu vhodnega toka
- `void close() throws IOException` – zapre bralnik

Primer uporabe branja iz std. vhoda:

```
BufferedReader br=new BufferedReader(new
                  InputStreamReader(System.in));
br.readLine(); // prebere vrstico iz std. vhoda
```

Strukturirano branje – Scanner (`java.util.Scanner`)

Razred `Scanner` omogoča strukturirano branje vhodnega toka (ali niza). `Scanner` razbije vhodni tok glede na poljubne razmejitvene znake oz. regularni izraz (privzeto je to prazen prostor) in omogoča enostavno branje besed, števil itn.
Nekatere metode:

- `Scanner(InputStream` *in*`)` – konstruktor, ki naredi nov objekt na podlagi podanega vh. toka. Ta je lahko npr. std. vhod; primer: `Scanner sc=new Scanner(System.in);`
- `Scanner(File` *in*`) throws FileNotFoundException` – konstruktor naredi nov objekt na podlagi podane datoteke: `Scanner sc=new Scanner(new File("t.txt"));`
- `Scanner(String` *str*`)` – konstruktor, ki naredi nov objekt na podlagi podanega niza *str*
- `void close()` – zapre scanner
- `boolean hasNext()` – vrne true, če na vhodu čaka nova vrednost, ki jo lahko preberemo
- `boolean hasNext`*X*`()` – vrne true, če na vhodu čaka nova vrednost tipa *X*, ki jo lahko preberemo. *X* je lahko osnovni tip (*Boolean*, *Byte*, *Double*, *Float*, *Int*, *Long*, *Short*) ali *Line*, ki predstavlja vrstico.
- `String next()` – vrne naslednjo vrednost z vhoda kot niz. Sproži `NoSuchElementException`, če na vhodu ni več vrednosti za branje
- *X* `next`*X*`()` – vrne naslednjo vrednost tipa *X* z vhoda (glej opis `hasNext`*X*). Sproži `InputMismatchException`, če vrednost ni tipa *X*, ali `NoSuchElementException`, če na vhodu ni več vrednosti za branje.
- `Scanner useDelimiter(String` *pattern*`)` – nastavi razmejitveni izraz na novo vrednost *pattern* (privzeto je to prazen prostor, Scanner torej privzeto razbija vhod na besede). Vrne samega sebe (`this`).

Pisanje – PrintWriter

Na poljuben tekstovni izhodni tok lahko pišemo preko metod razreda `PrintWriter`:

- `PrintWriter(Writer` *out*`)` – konstruktor, ki naredi nov objekt na podlagi podanega pisalnika. Ta je lahko npr. pisalnik v datoteko (`FileWriter`)
- `void print(String s)` – izpiše niz *s* na izhodni tok
- `void println(String s)` – izpiše niz *s* in znak za novo vrstico na izhodni tok
- `void print(`*Tip* `x), void println(`*Tip* `x)` – izpiše vrednost *x*, kjer je *Tip* poljuben tip (enostavni ali razred)
- `void close()` – zapre vhodni tok

Java programska kartica

Grafični uporabniški vmesnik (GUI)

AWT paket knjižnic za gradnjo GUI `(java.awt.*)`

Graphics je abstrakten razred, ki omogoča risanje na komponente, zato vsebuje vse osnovne metode za risanje. Nekaj metod:

- `void drawLine(int x1,int y1,int x2,int y2)` – nariše črto med točkama (*x1*,*y1*) in (*x2*,*y2*)
- `void drawRect(int x,int y,int w,int h)`/ `fillRect(x,y,w,h)` – nariše (zapolnjen) pravokotnik z gornjim levim ogliščem v točki (*x*,*y*), širino *w* in višino *h*
- `void drawOval(int x,int y,int w,int h)`/ `fillOval(x,y,w,h)` – nariše (zapolnjeno) elipso (ali krog) znotraj pravokotnika, ki ga določajo argumenti
- `void drawArc(int x,int y,int w,int h,int z, int kot)`/`fillArc(x,y,w,h,z,kot)` – nariše (zapolnjen) eliptični (ali krožni) izsek znotraj podanega pravokotnika, z začetkom pri *z* stopinjah in velikosti *kot* stopinj
- `void drawString(String niz,int x,int y)` – nariše besedilo *niz*, osnovna črta levega znaka je na (*x*,*y*)
- `boolean drawImage(Image img, int x, int y, ImageObserver obs)` – nariše sliko *img* na lokacijo *x*,*y*; obvešča objekt *obs*, ko se slika izrisuje. Vrne `true`, če je bila slika do konca izrisana.
- `void setColor(Color c)` – nastavi barvo risanja na *c*
- `void setFont(Font f)` – nastavi pisavo na *f*

Component je abstrakten razred, ki je nadrazred vseh AWT (in tudi Swing) gradnikov. Gradnik ima svojo grafično predstavitev, s katero lahko interaktira uporabnik.

- `void setSize(int w,int h)` – nastavi velikost gradnika na *w* in *h*
- `void setLocation(int x, int y)` – premakne gradnik na novo lokacijo, zgornji levi kot je na (*x*, *y*)
- `Graphics getGraphics()` – vrne grafični kontekst komponente, na katerega lahko rišemo
- `void repaint()` – ponovno izriše komponento

Container predstavlja vsebnik, ki vsebuje druge gradnike.

- `Component add(Component comp)` – v vsebnik doda gradnik *comp*. Vrne gradnik *comp*.

Swing paket knjižnic za gradnjo GUI `(javax.swing.*)`

JComponent je osnovni razred vseh Swing gradnikov, razen osnovnih vsebnikov. Nekatere metode (vse so `void`):

- `paintComponent(Graphics g)` – izriše gradnik; za risanje se lahko uporabi podani grafični kontekst *g*
- `setBackground(Color c)` – nastavi barvo ozadja na *c*
- `setForeground(Color c)` – nastavi barvo risanja na *c*

JFrame, **JDialog** in **JApplet** so posebni gradniki vsebniki, ki določajo samostojno okno in jih ne moremo vključevati v druge gradnike. Nekaj metod:

- `void setDefaultCloseOperation(int op)` – nastavi operacijo, ki se izvede, ko uporabnik zapre okno; privzeta vrednost je `HIDE_ON_CLOSE`; pri `JFrame` se za izhod iz aplikacije ponavadi uporabi metodo `exit` razreda `System`, ki je mapirana v vrednost `JFrame.EXIT_ON_CLOSE`
- `Container getContentPane()` – vrne referenco na delovno površino vsebnika
- `void setLayout(LayoutManager m)` – nastavi razporejevalnik gradnika na podani razporejevalnik *m*

JPanel je vmesni vsebnik, v katerega lahko polagamo gradnike ter omogoča združevanje gradnikov v skupine.

- `void setLayout(LayoutManager m)` – nastavi razporejevalnik na podani razporejevalnik *m*

JButton, **JTextField**, **JLabel** in podobni ostali gradniki ne morejo vsebovati drugih komponent in morajo biti vsebovani v katerem od vsebnikov.

- konstruktorji `JButton(String s)`, `JTextField(String s)` ali `JLabel(String s)` ustvarijo gradnik z besedilom *s*
- `void setText(String t)` – nastavi besedilo, izpisano v gradniku
- `String getText()` – vrne besedilo, izpisano v gradniku

Apleti `(javax.swing.JApplet)`

Glavne metode apleta:

- `init()` – brskalnik metodo kliče, ko je aplet naložen v sistem; vedno se kliče pred prvim klicem metode `start()`
- `start()` – kliče brskalnik ob začetku izvajanja apleta
- `stop()` – kliče brskalnik ob zaustavitvi izvajanja apleta
- `destroy()`–kliče brskalnik ob uničenju apleta s smetarjem
- `URL getDocumentBase()` – vrne URL strani z apletom
- `URL getCodeBase()` – vrne URL `.class` datoteke
- `String getParameter(String p)` – vrne vrednost parametra *p* (parametre podamo v HTML kodi)

Primer HTML kode, ki vključuje aplet:

```
<applet code="abc.class" height=300 width=300>
<param name=parameter1 value=vrednost1>
</applet>
```

Razporejevalniki `(java.awt.*)`

- `BorderLayout` razdeli vsebnik na pet regij, ki jih določajo konstante: `NORTH`, `SOUTH`, `EAST`, `WEST` in `CENTER`
- `FlowLayout` gradnike dodaja po vrsti od leve proti desni
- `GridLayout` razdeli vsebnik v mrežo enakih celic, njeno velikost (vodoravno, navpično) podamo v konstruktorju

Dogodki in poslušalci `(java.awt.event.*)`

Gradniki se odzivajo na dogodke s pomočjo poslušalcev, ki jih vežemo nanje. Poslušalce `KeyListener`, `MouseListener`, `MouseMotionListener` lahko vežemo na vsak Swing ali AWT gradnik; `ActionListener` vežemo na gumb, tekstovno polje in podobne gradnike. Metode za dodajanje poslušalca gradniku (vse so `void`):

- `addKeyListener(KeyListener l)` – gradniku doda poslušalca za dogodke tipkovnice
- `addMouseListener(MouseListener l)` – gradniku doda poslušalca za dogodke miške
- `addMouseMotionListener(MouseMotionListener l)` – gradniku doda poslušalca za premik miške
- `addActionListener(ActionListener l)` – doda poslušalca akcij gumbu, vnosnemu polju ...

Metoda vmesnika `ActionListener` (je `void`):

- `actionPerformed(ActionEvent e)` – se kliče ob akciji

Metode vmesnika `KeyListener` (vse so `void`):

- `keyPressed(KeyEvent e)` – se kliče ob pritisku tipke
- `keyReleased(KeyEvent e)` – se kliče ob sprostitvi tipke
- `keyTyped(KeyEvent e)` – se kliče ob pritisku tipke

Metode vmesnika `MouseListener` (vse so `void`):

- `mouseClicked(MouseEvent e)` – klik gumba
- `mouseEntered(MouseEvent e)` – vstop miške v gradnik
- `mouseExited(MouseEvent e)` – izstop miške iz gradnika
- `mousePressed(MouseEvent e)` – pritisk gumba miške

Java programska kartica

- `mouseReleased(MouseEvent e)` – sprostitev gumba miši

Metode vmesnika `MouseMotionListener` (vse so `void`):

- `mouseDragged(MouseEvent e)` – se kliče ob premiku miške s pritisnjenim gumbom
- `mouseMoved(MouseEvent e)` – se kliče ob premiku miške

Razred `EventObject` in iz njega izpeljani razredi `ActionEvent`, `KeyEvent` in `MouseEvent` imajo metodo:

- `Object getSource()` – vrne objekt, na katerem se je zgodil dogodek

Razred `KeyEvent`:

- `char getKeyChar()` – vrne znak (tipko) tega dogodka

Nekaj metod razreda `MouseEvent`:

- `int getButton()` – vrne gumb, ki je spremenil stanje
- `int getX()` – vrne *x* pozicijo, relativno glede na gradnik
- `int getY()` – vrne *y* pozicijo, relativno glede na gradnik

Podatkovne strukture `(java.util.*)`

`Collection<Tip>`

Osnovni vmesnik, ki ga implementirajo vse podatkovne strukture, ki shranjujejo posamezne elemente, med drugim: `ArrayList`, `Vector`, `Stack`, `Set`, `SortedSet` ...

- `boolean add(Tip el)` – doda *el* v zbirko in vrne `true`
- `void clear()` – zbriše vse elemente zbirke
- `boolean contains(Object o)` – vrne `true`, če zbirka vsebuje podani objekt *o*
- `Iterator<Tip> iterator()` – vrne objekt tipa `Iterator <Tip>`, ki omogoča sprehod po elementih zbirke
- `boolean remove(object o)` – zbriše element *o* in vrne `true`, če je brisanje uspešno
- `int size()` – vrne število elementov v zbirki

`Iterator<Tip>`

Osnovni vmesnik, ki ga implementirajo razredi, ki omogočajo sprehajanje po elementih zbirk. Metode:

- `boolean hasNext()` – vrne `true`, če obstaja naslednji element
- `Tip next()` – vrne naslednji element
- `void remove()` – zbriše trenutni element. Lahko se kliče le enkrat na vsak klic metode `next()`.

`ArrayList<Tip>`

Predstavlja polju podobno podatkovno strukturo, ki jo lahko dinamično povečujemo in zmanjšujemo z dodajanjem ali odvzemanjem elementov vrste `Tip`. Poleg metod vmesnika `Collection` vsebuje tudi metode:

- `boolean add(Tip el)` – doda *el* na konec in vrne `true`
- `void add(int i, Tip el)` – vrine element *el* na mesto *i*
- `Tip get(int i)` – vrne element na *i*-tem mestu
- `int indexOf(Object el)` – poišče prvo pojavitev *el* in vrne njegov indeks oz. -1, če elementa ne najde
- `Tip remove(int i)` – zbriše element na *i*-tem mestu
- `Tip set(int i, Tip el)` – zamenja element na *i*-tem mestu z elementom *el*

`HashSet<Tip>`

Predstavlja množico elementov tipa `Tip`. Poleg metod vmesnika `Collection`, vsebuje tudi:

- `boolean add(Tip el)` – doda *el*, če ga še ni v množici

`HashMap<TipK, TipV>`

Predstavlja podatkovno strukturo, ki shranjuje pare <*ključ*, *vrednost*>. Tip ključa in vrednosti določata `TipK` in `TipV`.

- `void clear()` – zbriše vse elemente
- `boolean containsKey(Object key)` – vrne `true`, če `HashMap` vsebuje podani ključ *key*
- `boolean containsValue(Object value)` – vrne `true`, če `HashMap` vsebuje podano vrednost *value*
- `TipV get(Object key)` – vrne vrednost tipa `TipV`, povezano s podanim ključem *key* ali `null`, če ključa ni
- `Set<TipK> keySet()` – vrne objekt tipa `Set<TipK>`, ki vsebuje vse ključe, shranjene v `HashMap`
- `TipV put(TipK key, TipV value)` – shrani par ključ *key*, vrednost *value* v `HashMap`
- `TipV remove(Object key)` – zbriše vrednost, shranjeno pod ključem *key* iz `HashMap`, in vrne to vrednost
- `int size()` – vrne število shranjenih elementov
- `Collection<TipV> values()` – vrne objekt, ki vsebuje vse vrednosti, shranjene v `HashMap`

Niti `(java.lang.Thread)`

Razred, v katerem želimo napisati kodo, ki bo tekla kot samostojna nit, mora implementirati vmesnik `Runnable` ali dedovati iz razreda `Thread`. Koda niti se mora nahajati v metodi `public void run()` tega razreda.

Nekatere metode razreda `Thread`:

- `Thread(Runnable target)` – konstruktor, ki mu kot parameter podamo objekt *target*, ki implementira vmesnik `Runnable` in ima posledično definirano metodo `run()`
- `static Thread currentThread()` – vrne referenco na nit, ki se trenutno izvaja
- `void interrupt()` – prekine to nit
- `void join(long m) throws InterruptedException` – čaka največ *m* milisekund, da se nit konča. Če je *m* enak 0, čaka neskončno časa, da se nit konča. Če se nit prekine, sproži izjemo.
- `void run()` – to metodo mora definirati razred, ki deduje iz `Thread`, vsebuje pa naj kodo, ki se bo izvajala v niti
- `static void sleep(long millis) throws InterruptedException` – povzroči, da trenutna nit preneha z izvajanjem za *millis* milisekund. Če se nit prekine, sproži izjemo.
- `void start()` – povzroči, da se nit začne izvajati – pokliče metodo `run()`
- `static void yield()` – začasno ustavi izvajanje trenutne niti, da se lahko izvajajo ostale niti

Koristne metode razreda `Object`:

- `void notify()` – zbudi eno od niti, ki čaka v vrsti tega objekta (preko metode `wait()`)
- `void notifyAll()` – zbudi vse niti, ki čakajo v vrsti tega objekta (preko metode `wait()`)
- `void wait(long t) throws InterruptedException` – povzroči, da nit čaka *t* milisekund (če je *t* enak 0, potem čaka neskončno časa) ali dokler neka druga nit ne pokliče `notify()` oz. `notifyAll()` na tem objektu

Sinhronizacija: če želimo, da določeno metodo hkrati izvaja le ena nit, jo lahko deklariramo kot `synchronized` in s tem preprečimo morebitne konflikte pri dostopu do skupnih virov.