1. *JAVA* ELEMENTAI

Šiame skyriuje pateikiama elementarių žinių apie programavimą *Java,* kad būtų galima suprasti paprasčiausius savarankiškų programų ir klientinių programų pavyzdžius.

1.1. Kalbos konstrukcijos

Kalbos konstrukcijos sudaromos iš leksemų – simbolių grupių, turinčių elementarią reikšmę:

- identifikatorių – klasėms, metodams, klasių laukams (ar kintamiesiems) įvardyti. Konstruojami iš mažųjų ir didžiųjų raidžių, skaitmenų, simbolių $ ir _ . Skaitmenys negali pradėti identifikatoriaus;
- raktinių žodžių;
- literalų – simbolių grupių duomenų reikšmėms perduoti (kitose kalbose jie dažnai vadinami „konstantomis“);
- operacijų ženklų;
- skirtukų – () [] {} ; , . ;
- komentarų – eilutiniai komentarai, pradedami nuo bet kurios eilutės vietos, rašomi po simbolių //. Kelių eilučių komentarai apimami tokiais komentarų skliaustais: /* *komentarų eilutės* */.

1.2. Objektų kūrimas ir darbas su jais

Java – „gryna“ objektiškai orientuoto programavimo (OOP) kalba. Pagrindinis kodo elementas – klasė; tiesiogine prasme programos arba modulio sąvokų kalboje nėra. Tačiau kartais paskaitų kurse visą kompiliavimui atiduodamą kodą (tikslus terminas – kompiliavimo vienetas) vadinsime programa.

Viskas, išskyrus primityviuosius duomenis, kode yra objektai: ir programuotojo sukurti nauji duomenų tipai – klasės, ir programos vykdymo metu generuojamos išimtys, ir programos vykdymo lygiagretūs srautai. Ir primityviesiems duomenims yra klasės-kevalai, kad būtų galima jais manipuliuoti kaip ir visais objektais.

Objekto identifikatorius iš esmės yra rodyklė (nuoroda) į objektą. Pavyzdžiui,

String s; – sukuriama rodyklė į *String* tipo objektą; ji nenutaikyta į jokį objektą, todėl turi reikšmę *null* ir jai negalima pasiųsti jokio pranešimo – bus deklaruota kompiliavimo klaida.

String s = new String(“abc”); – čia operatorius *new* sukuria ir patį objektą, ir nutaiko rodyklę į objekto atminties srities pradžią.

String s = “abc”; – šiam vidiniam *Java* duomenų tipui galimas toks supaprastinimas.

Visi objektai saugomi kompiuterio operatyviosios atminties srityje *heap*. Kompiliatorius nežino, nei kokie objektai bus sukurti kodo vykdymo metu, nei kiek jų bus. Atmintis objektams skiriama dinamiškai, kodo vykdymo metu operatoriaus – metodo-konstruktoriaus *new* pagalba. Tai – lanksti sistema, tačiau lėta.

Iš programos neįmanoma kreiptis į kompiuterio registrinę atmintį procesoriaus viduje ar į dėklinę atmintį (*stack*) operatyviojoje atmintyje. Tačiau kompiliatorius gali sudėti į dėklinę atmintį rodykles į objektus.

Jei objektas ar kuris nors jo laukas yra statinis – deklaruotas su modifikatoriumi *static*, jis dedamas į vadinamąją statinę operatyviosios atminties talpyklą – viso kodo vykdymo metu duomeniui skirtoji atmintis yra apibrėžtoje operatyviosios atminties vietoje.

1.3. Primityvieji duomenys

Java turi tokius primityviuosius duomenų tipus (pavadinimas, duomens ilgis, esminės charakteristikos, klasės-kevalo pavadinimas):

Loginiai	*boolean*	1B		*Boolean*
Simboliniai	*char*	2B	*Unicode* simboliai (žr. www.unicode.org)	*Character*
Aritmetiniai:			kitimo diapazonas	
sveikieji	*byte*	1B	–128 – +127	*Byte*
	short	2B	–32768 – +32767	*Short*
	int	4B	~–2*10e9 – ~+2*10e9	*Integer*
	long	8B	~–9*10e18 – ~+9*10e18	*Long*
realieji	*float*	4B	pagal IEEE754 standartą	*Float*
	double	8B	pagal IEEE754 standartą	*Double*

Visi aritmetiniai duomenys turi ženklą.

Standartas IEEE754, realizuotas daugelyje programavimo kalbų, nustato tokias realiųjų skaičių charakteristikas: viengubo tikslumo duomenų tikslumas yra 6–7 reikšminiai skaitmenys, kitimo diapazonas yra iki ~+/–10e38, mažiausia skirtinga nuo mašinos nulio reikšmė yra ~+/–10e–38. Dvigubo tikslumo duomenims analogiškos charakteristikos: 15–16 reikšminių skaitmenų, ~+/–10e308 ir ~+/–10e–308.

Atitinkamų tipų literalai užrašomi taip:

boolean	galimos reikšmės	*true* arba *false*
byte, short, int	pavyzdžiui,	*123* *0123* – duomuo aštuntainėje sistemoje *0x123* arba *0X123* – duomuo šešioliktainėje sistemoje
long	pavyzdžiui,	*123456789l* arba *123456789L*

float	pavyzdžiui,	*1.2f* arba *1.2F*, arba eksponentine forma *1.2e0f, 12e-1f* ir pan.
double	pavyzdžiui,	*1.2* arba *1.2d* ar *1.2D*, arba eksponentine forma *1.2e0* ir pan.
char	simbolių reikšmės apimamos ‘ ‘. Specialieji simboliai užrašomi taip: nauja eilutė – *\n* , ‘ – *\’* , \ – ** , tabuliatorius – *\t* ir t. t.	

Dažnai prie primityviųjų duomenų priskiriama ir *Java* vidinė klasė *String* - eilutinio tipo duomuo. Jos literalas užrašomas tarp angliškų kabučių simbolių “ “.

Taigi vieną simbolį kode galima saugoti ir kaip primityvųjį duomenį *char*, ir kaip klasės *Character* objektą:

```
char c = ‘a’;
Character cClass = new Character( ‘a’);
```

Java turi klases *BigInteger* ir *BigDecimal* su metodais, skirtais operuoti bet kokio tikslumo sveikaisiais ir realiaisiais skaičiais.

1.4. Identifikatorių matomumo sritys

Angliškai tai vadinama vienu žodžiu *scope*. Sritis apibrėžia tiek vardo matomumą, tiek ir jo gyvavimo laiką. Faktiškai čia aptariama matomumo sritis metoduose apibrėžtiems vardams; klasių kūnuose apibrėžtų vardų sritys paklūsta specialiems modifikatoriams (pavyzdžiui, *public* ar *private*), kurie bus aptarti vėliau.

Matomumo sritį apibrėžia skliaustai *{ }*.

Pavyzdžiui,

```
{ int x = 1;
    // čia x yra matomas ir prieinamas
        { int y = 2;
            // čia matomi x ir y
        }
   // čia matomas x, o y nematomas
}
  // čia nematomi nei x, nei y
```

Java neleidžia įdėtinėje vidinėje matomumo srityje naudoti to paties vardo. Pavyzdžiui, tai yra klaidingas fragmentas:

```
{ int x = 1;
        { int x = 2; }   }
```

Matomumo sritis objektams:

```
{  String s = new String( “abc” );  }
```

/ matomumo srities pabaiga. Čia nebeliks rodyklės s, tačiau objektas, į kurį anksčiau rodė rodyklė, išlieka. Jam skirta atmintis bus grąžinta operacinei sistemai tik po to, kai savo darbą atliks „šiukšlių rinktuvas“, periodiškai peržiūrintis kodui skirtą atmintį ir surenkantis atmintį objektų, į kuriuos nerodo jokia nuoroda. */*

1.5. Klasės

Klasė deklaruojama:

[modifikatorius]*class* AclassName *{* /* klasės kūnas */ *}*

Klasės objektas, pavyzdžiui, vardu „*c*“, sukuriamas metodu-konstruktoriumi:

AClassName c = new AClassName( );

Klasės kūne gali būti klasės kintamieji – laukai ir metodai. Sukūrus objektą, jo laukais operuojama taip:

```
class DataOnly {
  int i;
  float f;
}

. . .
DataOnly d = new  DataOnly( );
d.i = 1;
d.f = 1.2f;
. . .
```

Jei klasės laukas yra kitos klasės objektas, tai šio laukas išskiriamas toliau tęsiant „taškinę“ adresaciją:

d.i.a = . . .

ir t. t.

Jei klasės laukams – primityvams reikšmės nėra priskirtos, pagal nutylėjimą jiems suteikiamos nulinės reikšmės (*boolean* primityvui – reikšmė *false*). Tačiau lokaliems metoduose deklaruotiems laukams – primityvams reikšmės tokiu būdu nėra skiriamos.

Metodas deklaruojamas:

[modifikatorių sąrašas] grąžinamos reikšmės tipas|*void* vardas([argumentų sąrašas]) { /* metodo kūnas */ }

Metodo kūną užbaigia operatorius *return* [grąžinama reikšmė]. Kai metodas negrąžina jokios reikšmės (t. y. deklaruotas su raktažodžiu *void*), *return* nebūtinas. Metodas „*m*“ objektui „*o*“ kviečiamas arba taip:

```
o.m( );          // – kai metodas negrąžina jokios reikšmės;
```

arba turi būti kito operatoriaus dalimi, jei reikšmę, pavyzdžiui, *float* tipo, grąžina:

```
float f = o.m( );
```

1.6. Statiniai klasių laukai ir metodai

Jei kurios nors klasės visų objektų kuris nors laukas yra bendras visiems objektams ir turi saugoti tokią pačią reikšmę, jis deklaruojamas su modifikatoriumi *static*. Tokie laukai kitaip nei dinaminiai laukai saugomi atmintyje ir kitaip jiems priskiriamos pradinės reikšmės (išsamiau tai aiškinama kalbant apie metodų – konstruktorių atlikimo tvarką). Pavyzdžiui:

```
class StaticField {
        static int i = 1;
}
. . .
StaticField sf1 = new StaticField( ),
            sf2 = new StaticField( );
```

Dabar *sf1.i* ir *sf2.i* turi tokią pat reikšmę ir yra toje pat atminties srityje. Į tokį lauką galima kreiptis tiek objekto, tiek klasės vardu: *sf1.i* arba *StaticField.i* .

Panašiai yra su statiniais metodais: juos galima kviesti tiek konkrečiam objektui, tiek klasei:

```
class StaticMethod {
        static void incr( ) {
                StaticField.i++ ;
        }
}
. . .
StaticMethod sm = new StaticMethod( );
sm.incr( );
StaticMethod.incr( );  // alternatyvi kvietimo forma
```

1.7. Veiksmų operatoriai

Yra aritmetinių veiksmų operatoriai: +, – ,* , /, % (dalybos likutis).

Prieskyros operatoriai: =, += (prie esamos reikšmės pridėti; kitos prieskyros skirtos panašiems dalykams), –=, *=, /=, %= (esamą reikšmę pakeisti likučiu). Pavyzdžiui:

```
. . .
int i, j = 2;
i = (j+=2)*(j+3);
System.out.println( “j=”+j+”i=”+i ); // bus spausdinama j=4 i=28
```

. . .

Prieaugio ir mažinimo veiksmų operatoriai: ++, – –. Abiem operatoriams yra *pre-* ir *post-* formos. Gali būti taikomi bet kuriam (tik ne *boolean* tipo) primityvui: sveikiesiems duomenims veiksmo žingsnis yra 1, realiesiems – 1.f ir 1.d, simboliniams – vienas simbolis pagal simbolių eilės tvarką. Pavyzdžiui:

```
. . .
int i, j, k;
i = 1;
j = i++;  // j=1: pirma atliekama prieskyra, paskui – prieaugio operacija
          //  (post- forma)
k = ++i; // k=3 – prieaugio pre- forma
. . .
```

Santykio veiksmų operatoriai: = =, *!=*, >, <, >=, <=.

Loginiai operatoriai: && (loginė daugyba), || (loginė sudėtis). Veiksmai atliekami naudojant „trumpąją atlikimo schemą“: jei pirmojo veiksmo operando reikšmė nulemia viso veiksmo reikšmę, antrojo operando reikšmė nebeskaičiuojama.

Bitinio sulyginimo ir bitinio perslinkimo veiksmai. Likę *Java* nuo tų laikų, kai kalba buvo orientuojama elektronikos įrenginiams valdyti, todėl čia jų nenagrinėsime.

Visi veiksmai atliekami pagal veiksmų vyresnumo tvarką. Tačiau patogiau veiksmų atlikimo tvarką reguliuoti įrašant į reiškinius skliaustus ( ).

Programuojant aritmetinius reiškinius, būtina turėti galvoje ir vieną nemalonią *Java* ypatybę: jei įvyksta perpildymas, ši išimtinė situacija nėra deklaruojama, tik pakeičiamas duomens ženklas ir imama kitokia gana sudėtingai apibrėžiama duomens reikšmė.

1.8. Išvestis į pultą

Tam naudojamas metodas *println( ):*

System.out.println(String s);

Pavyzdžiai:

```
System.out.println( “Row Number” );          // arba
. . .
String  s1 = “Row”,
        s2 = “ Number”;
System.out.println( s1+s2 );  // konkatenacija – tekstinių duomenų sujungimas
. . .
int x = 3,
   y = 4;
System.out.println( x ); /* x automatiškai bus pervestas į String tipą (t. y.
                            automatiškai taikomas metodas toString( ).
                            Spausdins: 3 */
```

```
System.out.println( x+y ); /* iš pradžių bus suskaičiuotas aritmetinis reiškinys
                              (taip yra todėl, kad operatorius + jungia
                              aritmetinius duomenis ir yra interpretuojamas
                              kaip sudėties operatorius), tada gauta reikšmė bus
                              pervesta į String tipą. Spausdins: 7 */
System.out.println( "Sum of x and y is: "+x+y );  /* Spausdins:
                                Sum of x and y is: 34
                              Taip yra todėl, kad pirmasis operatorius + jungia
                              String tipo bei aritmetinį duomenis ir yra
                              interpretuojamas kaip sujungimo operatorius, o jo
                              darbo rezultatas bus String tipo reikšmė. Toliau
                              analogiška situacija pasikartos dar kartą, ir
                              galutinis rezultatas bus gautas toks, koks
                              parodytas aukščiau. Norint gauti atspausdintą
                              tikrą aritmetinio reiškinio rezultatą, aritmetinį
                              reiškinį reikia apskliausti: ...(x+y)...*/
```

Yra ir analogiškas metodas *print( )*, nelaužiantis eilučių:

```
System.out.print( "Ja" );
System.out.print( "va" );  // spausdins: Java
```

Specialieji simboliai išvestyje, pavyzdžiui, naujos eilutės simbolio \n veikimas:

```
System.out.println( "First row \n Second row" ); /*  spausdins:
                                                      First row
                                                      Second row  */
```

Analogiškai veikia ir kiti specialieji simboliai, esantys *String* tipo literaluose ar reiškiniuose.

1.9. Paprasčiausių autonomiškos programos ir klientinės programos pavyzdžiai

```
class FirstApplication{
        public static void main( String [ ] args ) {
                System.out.println( "The first application" );
        }
}
```

Ši programa į pulto ekraną išveda tekstą "*The first application*".

Programos paleidimo etapai:

1. Atspausdinus programos tekstą (pavyzdžiui, paprastu redaktoriumi *Notepad*), patalpinti programos tekstą į aplanką išorinėje atmintyje (patogiausia į atskirą, programai skirtą aplanką) vardu

FirstApplication.java. Kreipti dėmesį į didžiąsias ir mažąsias raides – *Java* kompiliatorius jas skiria. Vėliau matysime, kad, nepaskelbus startinės klasės (t. y. klasės, kurioje yra metodas *main*) vieša – *public*, rinkmenos vardą galima parinkti ir skirtingą nuo startinės klasės vardo.

2. *run* ir *command* komandomis paleisti pultą, *cd* komanda pereiti į aplanką, kuriame yra programos pradinė rinkmena, ir paleisti *Java* kompiliatorių *javac*:

javac FirstApplication.java

Sukompiliuota rinkmena (aišku, jei tik tekste nebuvo sintaksės klaidų) įrašoma į tą patį aplanką tuo pačiu vardu su plėtiniu *class*.

3. Programa paleidžiama kreipiantis į JVM interpretatorių *java* su startinės klasės vardu:

java FirstApplication

Pastabos. *Java* kodas susideda iš klasių. Čia vienintelėje kodo klasėje nėra paskelbta jokių laukų.

Įėjimo taškas į programą – metodas *main*, per argumentų sąrašą gaunantis eilutės tipo vienmatį masyvą *args* – programos paleidimo iš pulto komandinėje eilutėje po komandos *java FirstApplication* nurodytus eilutinius duomenis. Čia tokių duomenų nėra, masyvas lieka tuščias. Metodas turi būti *public* – viešas, statinis – *static* ir negrąžina jokios reikšmės – *void*.

Metode kreipiamasi į išvesties metodą *println*. Kreipiantis į *Java* vidinius metodus, reikia žinoti jų vietą *Java* vardų erdvėje. Pilnas *println* vardas yra *java.lang.System.out.println*. Paketo, kuriame saugomi sisteminiai *Java* dalykai, – *java.lang* vardo nurodyti nebūtina. Šis paketas automatiškai importuojamas į visas programas. Šiame pakete yra klasė *System*, turinti *PrintStream* klasės tipo lauką *out*, o klasė *PrintStream* turi statinį metodą *println*, todėl dažniausiai jis, kaip čia, kviečiamas klasės vardu.

Lygiai tą patį atliekančios klientinės programos (angl. *applet*) pavyzdys:

```
import javax.swing.*;
public class FirstApplet extends JApplet{
   public void init(){
      System.out.println( “The first applet” );
   }
}
```

Tai „dirbtinė“ klientinė programa, nes tokios programos yra skirtos ne spausdinti į pulto ekraną, o naršyklės rodomam HTML puslapiui praturtinti. Spausdinimas į pultą čia būtų pateisinamas nebent derinimo stadijoje, šiaip jau reikėtų pasitelkti iš klientinės programos superklasės paveldimus metodus grafiniams elementams išvesti.

Klientinės programos paleidimo etapai:

1. Įrašyti programos tekstą vardu *First Applet.java* į atskirą aplanką.

2. Sukompiliuoti tekstą komanda *javac FirstApplet.java*.
3. Į tą patį aplanką įrašyti HTML rinkmeną, kurioje kreipiamasi į klientinę programą:

```
<HTML>
<HEAD>
<TITLE> APPLET FirstApplet</TITLE>
</HEAD>
<BODY>
<APPLET   CODE = FirstApplet WIDTH = 100 HEIGHT =100 >
</APPLET>
</BODY>
</HTML>
```

   HTML kalba neskiria didžiųjų ir mažųjų raidžių, nekreipia dėmesio į intervalus. Kalba susideda iš sakinių (*tags*, pažodžiui – etiketės), kurie pradedami sakinio pavadinimu skliaustuose <…> ir baigiami tokiu pat pavadinimu skliaustuose </…>. Kai kurie sakiniai, kaip *APPLET*, yra konteineriai, todėl į juos galima įrašyti parametrus (*APPLET* – parametrą *CODE* – nurodantį klientinės programos vardą, jei reikia – su pilnu keliu; *WIDTH* – programos lango plotį vaizdo taškais, o *HEIGHT* – aukštį). Parametrų reikšmės kartais apimamos kabutėmis “”. Sakiniai *HEAD* ir *TITLE* sudaro titulinę puslapio dalį, o *BODY* aprašo patį puslapį.
   Įrašyti sukurtą HTML dokumentą, patogiausia – į tą patį aplanką.
4. Atiduoti HTML dokumentą naršyklei.

Yra specialus įrankis klientinėms programoms testuoti *appletviewer*, leidžiantis įvykdyti programas be naršyklės pagalbos. Įrankis reaguoja tik į HTML sakinį *APPLET*, todėl tokį sakinį galima komentaro pavidalu įrašyti tiesiog programos tekste ir nereikės rašyti atskiro HTML dokumento. Naudojant įrankį, programos tekstą reikės pakeisti į

```
//<APPLET  CODE  =  FirstApplet  WIDTH=100  HEIGHT=100>
//</APPLET>

import javax.swing.*;
public class FirstApplet extends JApplet{
   public void init(){
      System.out.println( ‘The first applet” );
   }
}
```

o 4-ajame etape vietoje naršyklės paleisti

```
appletviewer FirstApplet.java
```

Tokiu būdu paprastumo dėlei paleisime klientines programas viso kurso metu.

Pastabos. Klientinės programos gyvavimui palaikyti būtinos *Java* klasės, esančios paketuose *java.awt* (laikoma, kad dalis šių klasių jau yra pasenę ir

nenaudotinos) arba *javax.swing* (rekomenduojama naudoti klases iš šio paketo). Visi paketai, kurių vardai prasideda *javax* (nuo *Java extension*), turi *Java* galimybes praplečiančias klases. Paketas importuojamas operatorium *import*, o „*“ reiškia, kad dinamiškai importuojamos tik reikiamos įkelti klasės.

Klientinės programos startinė klasė (turinti metodą *init*) privalo būti paskelbta *public* ir turi paveldėti (*extends*) klientinės programos superklasės *JApplet* (jei būtų nutarta rinktis analogišką klasę iš *java.awt* paketo – tektų imti *Applet*) savybes ir metodus.

Įėjimo į programą taškas – metodas *init*, negrąžinantis jokios reikšmės.