Лекция 10

1. Масиви Java.

За разлика от C и C++, масивите в езика Java са обекти с изпълнимо представяне. Може да се декларира и заделя памет за масиви от всякакъв тип, както и да се декларират масиви от масиви, за да се получат многомерни масиви. Един масив може да се декларира по следният начин:

```
Dtype myDate[] ;
```

Този програмен код указва, че `myDate` е един неинициализиран масив от елементи от типа `Dtype` данни . На този етап единствената заделена памет за масива `myDate` е за идентификатора на масива. В някой момент от време при изпълнение на програмата в която се намира горната декларация ще бъде заделена памет за масива, кат за целта се използва специален оператор:

```
MyDate = new Dtype[20] ;
```

В случая се заделя памет за един масив от 20 елементи от клас `MyDate`, които се инициализират с празният елемент (null).

Достъпът до елементите на масивите се осъществява чрез стандартния C – стил на индексиране - елементите се номерират с последователни цели числа от **0** до **N-1**, където N е броят елементи в масива. За разлика от C и C++, в Java всички обръщения към елементи от масивите се проверяват, за да се осигури попадение на индексите в границите на масивите. Ако индексът е извън границите на масива се генерира изключение (exception).

За определяне на дължината на масив (броя елементи) се използва метод на класа **length()**, който дефинира масива. Например, обръщението към функцията (метода) `myDate.length(),` връща като стойност броя на елементите в масива `myDate:`

```
HowMany = myDate.length() ;
```

и целочислената променлива `HowMany` ще получи стойност 20.

В Java е премахната възможността за деклариране на указатели към масиви от елементи, действията с които създават предпоставки за несигурни операции. Отнета е възможността от излизане извън границите на масиви, което винаги води до неконтролируеми последствия за работата на програмата.

- **Символни (текстови) низове**. В езика Java, символните низове са обекти, а не масиви от символи, както е в езика C и C++. Има два различни класа за деклариране на символни низове – класа **String** е само за неизменни (read-only) обекти, а класа **StringBuffer** е за символни низове, които могат да бъдат модифицирани.

Въпреки, че символните низове са обекти от езика Java, компилаторът на Java следва традициите на C++, предоставяйки синтактични улеснения като разглежда низовете от символи затворени в двойни кавички като обекти от клас **String.** Следователно декларацията:

```
String hello = "Hello world"
```

декларира обект на клас **String** и го инициализира със символния низ, който съдържа символното представяне в Unicode на "Hello world".

Езикът Java разширява значението на оператора +, като включва и конкатенацията на символни низове. Това значи че е допустимо използването на програмни редове от типа:

```
System.out.println("Във разглежданият текст има" + num + "символа") ;
```

Този програмен фрагмент конкатенира (съединява) символния низ `"Във разглежданият текст има"` с резултата от преобразуване на числовата стойност на num в символен низ и след това добавя резултата със символния низ `"символа".` Както и при масивите, класовете **String** и **StringBuffer** предоставят метода length(), който предоставя броя на символите в символният низ.

Многомерни масиви. Многомерните масиви в Java представляват масиви от масиви Най-простата форма на многомерен масив е двумерния масив. Двумерният масив представлява

съвкупност от едномерни масиви. Двумерните масиви най-често представят данни които са в табличен или матричен вид. Ето как се декларира двумерен масив от цели числа:

```
int matrix [ ][ ] = new int[ ][ ] ;
```

За разлика от други езици в които се използват запетаи за отделяне на размерностите, Java поставя всяка размерност в отделни квадратни скоби. Обръщението към елемент от многомерен масив се извършва като индексът за всяка размерност се поставя в отделни скоби:

```
matrix [2][5] = 24 ;
```

Неравномерни масиви. В Java многомерните масиви могат да бъдат с различна дължина в някои от размерностите. Когато се заделя памет за многомерен масив, се посочва паметта само за първата размерност (най-лявата). Другите размерности могат да се задават допълнително. Тогава е възможно да се зададе различен брой елементи за друга размерност:

```
int matrix [ ][ ] = new int [5][ ] ;
matrix [0] = new int [4] ;
matrix[1] = new int[5] ;
matrix[2] = new int [5] ;
matrix[3] = new int [6] ;
matrix[4] = new int [6] ;
```

Използването на неравномерни масиви не се препоръчва за често използване, защото се усложнява логиката на алгоритмите за обработка на информацията. Но за някои специфични приложения те могат да се използват доста ефективно.

Инициализиране на масиви. Един многомерен масив може да бъде инициализиран, като инициализационният списък за всяка размерност се огради с фигурни скоби:

```
int matrix [ ][ ] = { {2,3,4,5},
                      {3,4,5,6},
                      {4,5,6,7},
                      {5,6,7,8}};
```

2. Аплети и събития

Аплетите използват елементи от архитектурата на машините и специална програмни техники. Една от тези техники е обработката на събития, която е елемент от използването на графични операционни среди. Събитията са елементите посредством които аплетите получават входна информация от окръжаващата програмна среда.

Аплетите се различават съществено по типа на програмиране от представените до тук примери. Те са малки програми, предназначени за предаване по Internet, и се изпълняват в среда на браузър. Тъй като виртуалната машина на Java е отговорна за изпълнение на Java програмите, включително и аплетите, то аплетите предлагат сигурен начин за динамично изпълнение на програми във Web. Програмна реализация на един прост аплет изглежда по следния начин:

```
// Елементарен аплет
import java.awt.* ;
import java.applet.* ;

public class TestApplet extends Applet {
    public void paint (Graphics g) {
        g.drawstring (“Това е елементарен аплет !”, 10,30) ;
        }
}
```

Аплетите започват с две конструкции **impor**t. Първата импортира класовете дефинирани в пакета Abstract Window Toolkit (AWT). Аплетите взаимодействат с потребителя посредством AWT, а не чрез базираните на конзолно управление I/O класове. AWT включва графичен интерфейс използващ прозорци за реализация на входно-изходни операции. Този пакет от

класове е доста голям и сложен, но като начало могат да се използват само най необходимите класове и методи дефинирани в него и постепенно да се разширява използването му.
Втората конструкция импортира пакета от класове **applet**. Този пакет съдържа класа **Applet**. Всеки новосъздаден аплет трябва да с наследник на **Applet**. Като разширение на него се декларира клас TestApplet, който управлява новосъздавния аплет. Той се декларира като public, тъй като до него трябва да има достъп от външен код.
В класа TestApplet е деклариран метод **paint ()**. Той се дефинира от класа **Component** на AWT (надклас на Applet) трябва да се предефинира в аплета. Методът **paint ()** има един параметър от тип **Graphics**. Този параметър съдържа описание на графичната среда, която се изпълнява аплетът.
В метода **paint ()** има извикване на **drawString ()**, който е член на класа Graphics. Този метод извежда текстов низ, започващ от указаното местоположение с координати X и Y.Методът е **drawString ()** аналогичен на **println ()** при конзолния начин на извеждане на информацията. Горният аплет ще изведе в специален прозорец за аплета текста "Това е елементарен аплет !".
Трябва да се отбележи, че аплетът няма метод **main ()**. Това е така, защото аплетите не са самостоятелни приложения, а се изпълняват от браузър или друга съвместима с аплети програма.
За да се изпълни един аплет в Web браузър, той трябва да бъде компилиран по същия начин както обикновените Java програми. След компилацията се получава байт код на програмата, а компилаторът записва този код във файл с същото наименование, както изходния файл и с разширение '.class'. Байт кода на компилиран аплет може да бъде включен в Web страница с помощта на HTML команда:

```
<applet code="TestApplet" width=250 height=300>  </applet>
```

<applet> е таг за включване на аплети в HTML, а параметрите **width** и **height** задават областта (размерите на прозореца) в който се показва съдържанието на аплета.
Аплетът има графичен интерфейс, който се реализира на клиентския компютър. Той може да се използва за достъп до ресурси, но само на такива, намиращи се на Web сървъра (хоста). Аплетът няма достъп до файловете на компютъра, на който е свален.
Размерът на прозореца в който се изпълнява аплета се задава от HTML кода, а не от програмния код на Java. За разлика от приложенията, аплетите се съхраняват на отдалечени компютри (в мрежата). При заявка те се зареждат в паметта на клиентския компютър и се свързват с локалните ресурси чрез браузъра. Чрез аплетите могат да се изпълняват проверки и изчисления на клиентския компютър.

3. Други възможности на Java.
Представената дотук информация за език Java, описва основните правила и възможности на езика. Java е съвременен и мощен език за програмиране, затова той съдържа много повече възможности, които обикновено се усвояват след добиване на определена практика в областта на програмирането. За целта трябва да се използва някое пълно ръководство за Java или подходящ справочник по програмиране. Тук ще бъде направен преглед на по-важните не описани в тази книга възможности на Java.

Абстрактни методи и класове. Абстрактните методи са мощна конструкция на обектно-ориентираното програмиране. Абстрактен клас представлява клас в който се дефинират методи, които реално не се реализират в него. Чрез въвеждането им се запазва място в класа, и при дефинирането на подкласове се прави пълно описание на тези методи. По този начин, може да се дефинира някакъв супер клас, на основата на който да се създадат множество подкласове за използване за конкретна програма.
Абстрактен метод може да се въведе в горния пример, в класа TwoDimFig. Абстрактен метод може да се дефинира за определяне на площа на фигурата:

```
abstract double Area () ;
```

Както се вижда, в родителския клас методът Area няма тяло и списък с аргументи. За деклариране му се използва ключовата дума **abstract**. Клас, който съдържа един или повече абстрактни методи, също трябва да се декларира, като абстрактен:

```
abstract class TwoDimFig {
```

Истинската декларация на абстрактния метод се извършва в съответните подкласове. Там се задава списъкът с аргументите и тялото на метода.
Когато един подклас наследява абстрактен клас, той трябва да имплементира (да бъдат дефинирани) всички абстрактни методи. Ако бъде пропуснат някои от абстрактните методи при описанието на подклас той ще се разглежда като абстрактен. В този случай той не може да се използва за обявяване на обекти. При опит за използване на такава декларация компилаторът ще генерира съобщение за грешка.
Използването на абстрактен клас може да подобри програмната реализация на конкретни приложения. В разгледания пример за двудименсионна геометрична фигура в родителския клас няма смислена концепция за лицето на недефинирана фигура, поради което се въвежда абстрактен метод. Това обаче изисква при задаване на всяка конкретна фигура (подклас) е задължително декларирането на метод за определяне на площа на фигурата.

Многопоточност в език Java. Съществуват два способа за реализиране на многозадачна работа на компютърните системи: базирана на процеси и на потоци (нишки). Базираната на процеси многозадачност се реализира обикновено от операционната система, която разделя времето на работа на микропроцесора на интервали (кванти) и го разпределя за отделните програми. При базираната на потоци (нишки) многозадачност една програма може да изпълнява няколко задачи едновременно. За целта се декларират отделни нишки като елемент с управляем код. Например, един текстов редактор може да форматира даден текст и същевременно да печата стига тези две дейности да са организирани в два отделни потока (нишки).
Основното предимство на многопоточността е, че се дава възможност за писане на ефективни програми, при които се използва бездействието на централните устройства за изпълнение на паралелни процеси. Заедно с многопоточността при работа на програмите, съществена роля играе и синхронизиране на работата на отделните потоци. Java притежава стабилна система за синхронизация на потоците.
Системата за многопоточност в Java се изгражда върху класа **Thread** и придружаващия го интерфейс **Runnable**. **Thread** капсулира един поток (нишка) на изпълнение. За да се създаде нов поток, програмата трябва да наследи **Tread,** или да имплементира **Runnable**.

Пакети. Пакетите са групи от свързани класове. Те подпомагат организацията на програмиране и създават още един слой на капсулиране (сигурност) на класовете и обектите. Пакетите осигуряват механизъм, чрез който свързаните части на програмите могат да се организират в един модул. Достъпът до класовете, дефинирани в един пакет се извършва чрез името на пакета. Тъй като пакетите съдържат взаимосвързани класове в Java са дефинирани специални права за достъп до кода в пакета. В пакета може да се дефинира код, който е достъпен за методи от класове в пакета, но да не е достъпен за код извън пакета.
Когато се дефинира даден клас за неговото наименование се заделя име от пространството от имен (namespace). Пространството от имена дефинира областта на валидност на имената. В Java не може два класа да използват едно и също наименование в пространството от имена. Това не е проблем за кратки и компактни програми. В големите програмни приложения може да се окаже трудно намирането на уникално наименование на всеки от множеството класове, които се използват. Освен това трябва да се избягват проблеми при свързването на програмни части създадени от различни програмисти, работещи по един проект. Решаването на тези проблеми в Java се извършва посредством пакетите. Когато един клас се дефинира в

пакет, името на този пакет се прикрепва към името на всеки клас, като по този начин се избягват колизиите с имената на други класове, намиращи се в други пакети.
Тъй като пакетът обикновено съдържа взаимосвързани класове, Java дефинира специални права за достъп до кода в пакета. В пакета може да се дефинира код, който е достъпен за код в същия пакет, но за код извън пакета това може да стане само ако пакетът, класът в пакета и метода в класа са декларирани като ***public***. Това позволява създаването на самостоятелни групи класове, в които операциите са недостъпни отвън.
Всички класове в Java принадлежат на някакъв пакет. Когато не указан оператор **package**, който декларира пакет, се използва подразбиращият се пакет на Java. Пакетът по подразбиране няма име, което го прави прозрачен. Поради тази причина програмистите не трябва да се грижат за деклариране на пакет, когато разработват неголяма програма и нямат нужда от разделяне на програмата на отделни пакети.
За дефиниране на пакет се използва служебната дума ***package***:

```
package project1 ;
```

Интерфейси. Интерфейсът дефинира множество от методи, които трябва да бъдат описани в даден клас. Самият интерфейс не реализира никакви методи. Той е чисто логическа структура. Интерфейсите са подобни на абстрактните класове, но в тях не се включват полета и реализиращ код. Те указват какво трябва да се прави, но не и как. Интерфейсът може да се разглежда, като инструкция какво трябва да прави даден метод, за да си използва за определени цели. Как ще се реализира това действие е въпрос на реалния клас (метод), който използва интерфейса.
Интерфейс се декларира с помощта на ключовата дума interface. След като бъде дефиниран един интерфейс, той може да бъде реализиран от един или няколко класа. За реализиране на интерфейса се използва ключовата дума **implements**.

```
public interface numbers {
        int getNext () ;  // връща следващо число
        void reset () ;  // започва отначало
        void setStart (int x) ;  // задава начална стойност
}
```

Това е пример на интерфейс който дефинира три метода. За тях не е указано какво трябва да правят.Този интерфейс може да се реализира чрез различни класове. Ето един пример:

```
class ByTwos implements numbers {
    int start ;
    int val ;

    ByTwos () { start = 0 ; val = 0 ; }
    public int getNext () { val += 2 ; return val ; }
    public void reset () {start = 0 ; val = 0 ; }
    public void setStart (int x) { start = x ; val = x ; }
    }
```

Това е една реализация на интерфейса, чрез която може да се генерира поредица от числа, всяко от които е с две по-голямо от предишното – метод getNext (). Методът reset () започва редицата от нула, а методът setStart (int x) може да стартира процеса от кое да е число. Друг клас може да реализира интерфейса така, че да се генерира поредица числа с разлика например 4 (class ByFour). Разликата от горния клас ще бъде, че вместо оператор val += 2 ; трябва да се използва оператор val += 4 ;.

Обработка на изключенията. Изключението е грешка, която възниква при изпълнение на програмата. С помощта на системата на Java за обработка на изключенията, грешките могат да се обработват и да не се стига до неконтролирано напускане на програмата. В Java са дефинирани голям брой стандартни изключения, които могат да се използват за управление

на програмата при възникването им. Използването на системата за обработка на изключенията е много полезна в процеса на изчистване на програмата от грешки.

Файлова система на Java. Програмите в Java извършват входно-изходните операции посредством потоци. Потокът е абстракция за представяне на последователно подредена информация в процес на въвеждане или извеждане. Потокът се свързва с физическо устройство чрез входно-изходната система на Java. Всички потоци имат еднаква логическа структура, дори физическите устройства, към които са свързани да са различни. В Java са дефинирани два вида потоци: байтови и символни.
Четенето и записа на информация се извършва посредством файлове. В Java всички файлове са байтово ориентирани и са осигурени методи за четене и запис на байтове от и във файл. Организацията на данните във файла може да бъде последователна и с пряк достъп. За тези типове организация съществуват методи за достъп до файловете.