Лекция 7

Обектно програмиране. Програмен език Java

Индустрията за програмно осигуряване е преминала в развитието си през няколко етапа, които са свързани с внедряване на нови технологии и методики за разработка на програмни продукти. В средата на 80-те години голяма популярност получи едно ново направление в програмирането – изкуствен интелект. Тази еуфория не продължи дълго. Това се случи благодарение на много причини, но една от тях беше, че в индустрията на програмното осигуряване навлезе концепцията за обектно-ориентирано програмиране (ООП). За разлика от изкуствения интелект, ООП имаше точно определени параметри и преимущества и беше способно да даде на програмистите конкретни инструменти за създаване на мощни програмни продукти. То осигури реални средства за опростяване на разработките и за многократно използване на компоненти в приложенията. Много специалисти считат ООП за неефективна и водеща до увеличаване на разходите технология в сравнение с традиционните системи на програмиране, но тя се наложи бързо и вече се използва много ефективно.

Обектно-ориентираното програмиране превъзхожда процедурното програмиране "отгоре-надолу" по възможностите за създаване на многократно използваем код и за моделиране на ситуации от реалния свят. Появата на Microsoft Windows придаде още по-голяма тежест на основанието за преминаване към ООП, защото съчетанието на ООП и GUI (Graphic User Interface) даде големи възможности за улесняване на програмирането в графични среди. ООП опростява решаването на сложните задачи при програмиране и позволява развитието на програмните езици.

Разработчиците на програмно осигуряване бързо се преориентираха към работа с обектно-ориентирани езици, създадени на основата на обикновените програмни езици – C++ създаден на основата на C, Object Pascal създаден на основата на Pascal и други.

Основните концепции на обектно-ориентираното програмиране в тази книга се представят посредством програмния език ***Java***. Използва се този език, защото той се основава на широко разпространения програмен език C++ и така се въвеждат голяма част от основните програмни концепции и правила на езика C++. Наред с това езикът Java е много близък с най-новия продукт на Microsoft – езика C#, който е основа на новата програмна платформа .NET Framework.

Java е нов език за програмиране, постигнал широка популярност за невероятно кратък период от време. Той е доста по-лесен за програмиране от много езици, но е изключително мощен и работи на различни платформи (операционни системи и програмни среди).

В началото (1995 година) Sun Microsystems разработва език за управление на така наречените set-top кутии, използвани от кабелните оператори за комуникация с потребителите, а също така и за програмиране на различни битови устройства, като тостери, печки, дистанционни устройства и други. Първоначално той е наречен **Oak (дъб)**. За да се работи удобно, е било необходимо той да е малък (за лесен пренос на командите по кабелната мрежа), и преносим (да работи с различни платформи). Затова този език е създаден като производен на езика C++.

Както доста открития set-top кутиите бяха бързо забравени, но с развитието на Web, Sun разбират че имат език, който може да се окаже доста подходящ за работа във Web. За да направят успешна маркетингова кампания Sun дадоха на продукта ново име "Java". Java не е акроним, а дума. На шега някои го интерпретират като "Just Another Vague Acronym" (Още едно неясно съкращение).

Java е проектиран така, че да посрещне предизвикателствата при разработване на приложения за хетерогенни разпределени мрежови среди. Основното сред тези предизвикателства е създаването на сигурни приложения, които консумират минимум системни ресурси, могат да работят на различни софтуерни и хардуерни платформи, допускат развитие и т.н.

Java се появи в публичното пространство за първи път през 1996 година и предизвика много надежди за разрешаване на проблемите при програмиране в Internet. Сега тези надежди са доста поохладени, тъй като Java не успя да отговори на всички очаквания, но това е така, защото очакванията бяха прекалено големи. Java е просто език за програмиране и има всички ограничения валидни за програмните езици. Различното е, че той е един от най-добрите програмни езици. Той е малък, работещ на различни платформи и не е само за WEB, а и за създаване на стандартни програмни приложения.
Java наследява своя синтаксис от езика C, а обектният модел е заимстван от C++. Основавайки се на това богато наследство, Java предоставя на програмистите мощно, и добре систематизирано програмно обкръжение, което е заимствано от най-добрите програмни среди и добавя нови характеристики. Затова, преходът от програмиране със C и C++ към програмиране с Java е много лесен и безпроблемен.
Поради приликите между Java и C++, особено в областта на обектно ориентираното програмиране, доста изкушаващо е да се приема Java, като Internet версия на C++. Това обаче не е точно, тъй като Java притежава значителни практически и философски различия със C++. Най-явното доказателство за това е, че Java и C++ не са съвместими. Те са проектирани за решаване на различен тип задачи и вероятно ще съществуват съвместно още дълго време.

1. Основни характеристики на езика Java

Основните характерни особености, които правят езика Java толкова популярен са:

- **Простота, обектна-ориентираност и познатост**. Простотата на езика позволява на програмисти с неголяма практика да работят със съвременни софтуерни технологии. За програмен език той е доста лесен за научаване. Основните концепции на езика се разбират лесно и бързо и позволяват на програмистите да създават смислени и полезни приложения още в периода на усвояването му.

Java е изцяло обектно-ориентиран (на най-ниско ниво). Обектно-ориентираната методология е неразделна част от Java и всички Java програми в някаква степен са обектно-ориентирани. Нуждите на разпределените клиент-сървър базирани системи съвпадат с капсулираните, обработващи парадигми на обектно-ориентирания софтуер. Java е чудесен начин да се започне изучаването на обектно ориентираните концепции на програмиране, без излишните усложнения въведени в C++.
Програмистите използващи Java разполагат с готови библиотеки с проверени обекти, които им предоставят голям функционален диапазон – структури от данни, входно-изходни и мрежови интерфейси, средства за създаване на графични елементи и други. Тези библиотеки могат да бъдат разширявани и обновявани.
Въпреки че езикът C++ е упрекван много за сложните синтактични конструкции, външната прилика на Java със C++ води до чувство на познатост на езика. Премахнатите излишни усложнения от C++ водят до улесняване на програмирането.

- **Смисленост и сигурност**. Java е създаден за разработване на софтуер, който да е много сигурен. За целта се извършват много проверки по време на компилация на програмите, както и по време на изпълнение на програмите.

Реализираният модел за управление на паметта без използване на указатели, премахва цели класове от типични грешки на програмистите използващи стандартните C и C++. С Java се разработва програмен код, за който е сигурно, че системата ще открие много бързо допуснатите грешки и основните проблеми няма да останат скрити до продажбата или внедряването на софтуерните продукти.
Java е предназначен за работа в разпределени среди, което значи че неговата сигурност е от първостепенно значение. За тази цел и изпълняващата система на Java е разработена с предвидени изключителни мерки за защита на програмният код на приложението от външни въздействия (вируси, хакерски достъп).

- независимост от архитектурата (преносимост). Java е проектиран за създаване на приложения, които работят в хетерогенни мрежови среди. В такива среди, приложенията трябва да са в състояние да се изпълняват на много хардуерни архитектури, операционни системи и потребителски графични интерфейси. За да се приспособи към разнообразните среди, компилаторът на Java генерира байт-код (bytecodes) – един независим от архитектурата междинен формат, който е проектиран за ефективно прехвърляне на приложения на различни хардуерни и софтуерни платформи. Този байт-код след това се изпълнява като се интерпретира (преобразува се в машинни команди и се изпълнява в реално време. Интерпретируемата природа на Java решава едновременно както проблема с разпространението на двоичния код, така и проблема с версиите – един и същ байт код работи на различни платформи.

Независимостта от архитектурата е само част от истинската преносимост. Със своите строги и точни дефиниции в синтаксиса на езика и структурата на типове данните, Java осигурява еднаква работа на приложенията на различните платформи. Тук не може да се получи несъвместимост на типовете данни при различни хардуерни и софтуерни платформи.

Независимостта от архитектурата и преносимостта на Java се осигуряват от Java Virtual Machine (Виртуална машина на Java). Това е спецификация на една абстрактна машина, за която компилаторите на Java генерират кода (байткода). Специфичните реализации на виртуалната машина на Java за дадени хардуерни и софтуерни платформи дава възможност да се стартират всички приложения написани на Java. Виртуалната машина на Java се основава на спецификацията на POSIX-интерфейс – една индустриална дефиниция на стандарт за преносим системен интерфейс. Реализирането на виртуалната машина на Java за различни архитектури е стандартна задача, тъй като платформата поема поддръжката на основните изисквания.

- **Висока производителност**. Производителността играе важна роля за софтуерните приложения. Java постига голяма производителност, тъй като интерпретатора работи с голяма скорост, без да извършва проверки на работната средата на приложението. Модулът за автоматично събиране на остатъци (garbage collector) работи като поток с нисък приоритет, и по този начин осигурява висока вероятност за достъпност на паметта когато потрябва и същевременно се повишава производителността. Приложенията, които изискват голяма изчислителна мощност могат да бъдат проектирани така, че модулите с които се извършват големи изчисления да бъдат написани на естествен машинен код и след това свързани със средата на Java. Когато приложението има интерактивен характер (работа в диалогов режим) режим на интерпретиране е достатъчно бърз.

- **Интерпретируемост, многопоточност и динамичност**. Интерпретаторът на Java може да изпълнява директно байт-код на всяка машина, на която е инсталирана виртуална машина на Java. В интерпретируема среда, фазата на свързване в цикъла на разработката на приложението е проста и лесна. Това осигурява по-кратък период за разработване на приложенията за разлика от традиционните езици, които изискват програмиране, компилиране, свързване, тестване.

Модерните мрежови приложения, обикновено изпълняват няколко дейности едновременно. Например, могат да се стартират няколко приложения, докато се тегли някакво изображение по Internet. Многопоточните възможности на Java осигуряват възможност да се разработват приложения с много конкурентни потоци на активност. По този начин се реализира висока степен на интерактивност.

Java подържа многопоточност на езиково ниво, като използва специално примитиви за синхронизация. Библиотеката на езика предоставя специален клас Thread, а работната среда – монитори и примитиви за условно блокиране. Системните библиотеки на Java са безопасни от гледна точка на многопоточност – те са достъпни без конфликти, дори при голямо количество конкурентни потоци за изпълнение.

Докато компилаторът на Java е строг при статичните проверки, езикът и работната среда са динамични по време на фазата на свързване. Класовете се свързват само при необходимост. Нови модули могат да бъдат свързани при поискване от всякъде, дори и чрез мрежата.

2. Синтактични правила на език Java.

Премахването на много от възможностите със съмнителна стойност на C++, правят езика Java относително малък и опростяват значително създаването на програмни приложения. Освен това, той изглежда познат на голяма част от програмистите занимаващи се с програмиране (тези които познават C и C++). Простотата на езика може да се илюстрира посредством традиционната за въвеждане в програмните езици програма HelloWorld. Написана на език Java, програмата изглежда по следния начин:

```
Class HelloWorld{        //Декларация на клас
     static public void main ( String args[]) {
          System.out.println ("Hello World !!! ");
     }
}
```

Първият ред на програмата съдържа ключовата дума от езика Java **class,** с която се декларира нов клас **`HelloWorld.`** Тъй като Java е изцяло обектен език, класът е основна единица за работа в езика. **`HelloWorld`** е името на класа. Дефиницията на класа започва с отваряща фигурна скоба и завършва със затваряща фигурна скоба (последната скоба в горния фрагмент). Елементите зададени между тези две скоби се наричат членове на класа. В обектно-ориентираните езици всяка подпрограма или функция се нарича метод и се отнася към някакъв клас или обект. В класа **`HelloWorld`** се съдържа само един метод наречен **main** (). Този метод има няколко характеристики, които са описани в реда от програмата съдържаща **main** (). Точното значение на думите, които описват тези характеристики ще бъде уточнено после, но тъй като те се срещат в по-голяма част от примерите ще бъде дадено известно пояснение.

Ключовата дума **`public`** е спецификатор за достъп. Той определя как другите части на програмата могат да се обръщат към членове на този клас. Ако един член на класа е обявен като **`public`**, той е достъпен за кода извън класа. В случая методът **main(),** трябва да се декларира като **`public`**, тъй като трябва да бъде извикан от код извън класа при стартиране на програмата. Обратният на **`public`** спецификатор за достъп е **`private`**, който не позволява достъп до члена от код извън декларирания клас.

Ключовата дума **`static`** позволява **main ()** да бъде извикан преди създаването на обект от този клас. Това е необходимо, тъй като **main ()** се извиква от Java интерпретатора, преди да създадени каквито и да са обекти.

Ключовата дума **`void`** указва на компилатора, че **main ()** не връща резултат. Както е известно, подпрограмите и функциите могат да връщат резултат и той има определен тип. Следователно, вместо void може да се използва какъвто и да е друг спецификатор за тип на връщания резултат.

Методът **main ()** е метод който се извиква в началото на всяко Java приложение. Всяка информация, която трябва да се предаде на метода се описва чрез променливите , зададени в скобите след наименованието на метода. В **main ()** има само един параметър String args[], който декларира параметър с име args и е масив от тип String. Тъй като **main ()** е стартиращ метод, той получава наличните при стартирането на програмата аргументи от командния ред на извикващата програма.

Методът **main ()** има едно обръщение към метод, който да изведе на стандартно устройство символния низ "Hello World". Програмният ред, който извежда низа "Hello World", извиква вградения метод **println** на обекта **out**. Обектът **out** е променлива от тип клас на класа **System** и реализира изходна файлова операция. Класът **System** е предварително дефиниран

клас, който осигурява достъп до системата, а **out** дефинира изходен поток, свързан с конзолата. По този начин **System.out** е обект, който дефинира изход към конзолата.
Както може да се предположи, изход към конзолата (текстов прозорец за съобщения на DOS) не се използва често в Java програмите и Java аплетите. Сега по-често се използват графични операционни системи и изход (вход) към конзолата се използва само за прости помощни програми, както и в примерите при изучаване на даден език.
В горния фрагмент е използван коментар (`//Декларация на клас`), който представлява пояснителен текст и се игнорира от интерпретатора на езика. Коментарите в Java са организирани както в език C++. Едноредовите коментари започват с две наклонени черти '//' и целият текст след тези символи се разглежда като коментар (игнорира се от интерпретатора). Коментар може да бъде разположен и върху повече от един ред, но тогава се използва отваряща комбинация от символи '/*' (наклонена черта и звездичка) и затваряща комбинация '*/'. Текстът, намиращ се между отварящата и затварящата комбинация символи се разглежда като коментар от Java.
В разглеждания пример, конструкцията **println ()**, завършва с точка и запетая ';'. Всички оператори в Java завършват с този символ. Това позволява един оператор (синтактическа конструкция) да се записва на повече от един ред, както и повече от един оператор да се разполагат на един ред.
За повечето програмни езици името на файла съдържащ изгодната програма (source code) може да бъде произволно. В Java случаят не е такъв. В Java ***името на файла, който се дава при записа на изходния текст е много важно***. Разширението на файла трябва да бъде '.java'. Например, горният текст на програмата HelloWorld може да бъде записан в един файл с наименование 'HelloWorld.java'. Разширението на файла съдържа четири символа, което означава, че операционната система трябва да поддържа дълги имена. Следователно, операционните системи DOS и Windows 3.1 не поддържат Java.
В Java един файл с изходен текст се нарича компилационна единица (compilation unit). Това е текстов файл, който съдържа една или повече дефиниции на класове. В Java съществува конвенция ***името на файла с изходния текст да съвпада с името на класа в който е дефиниран метода main()***. След компилация на файла с изходния текст, компилаторът създава файл с разширение '.class', който съдържа байткод на програмата. Байткодът не е изпълним код. Той трябва да се изпълни от виртуалната машина на Java.
За да се компилира файлът съдържащ кода на език Java трябва да е инсталиран пакет за разработки на Java (JDK – Java Developer Kit). Той съдържа виртуалната машина на Java. Трябва също така да е указан пътят до програмния пакет чрез променливата на Windows PATH. Например, ако е инсталирана версия на JDK 1.4.2 в основната директория на диск C, на променливата PATH трябва да се зададе стойност "C:\j2sdk1.4.2\bin\". Стойността на тази променлива може да се зададе посредством Control Panel/System/Advanced/Environment Variables.
Компилация на програмата на Java може да се извърши с команда зададена от командния ред в конзолен прозорец (DOS Prompt прозорец):

```
C:> javac Hello World.java
```

С тази команда се стартира интерпретаторът на Java – **javac**, намиращ се в директория ***bin*** на пакета за разработки на Java. Той създава файл Hello World.class, който съдържа байткод на зададената програма. Този файл може да се стартира за изпълнение посредством командата:

```
C:> java Hello World.
```

Java съдържа и клсове за създаване на приложения с графичен интерфейс, които ще бъдат разгледани по-късно. Конзолните приложения се използват по-често за учебни цели, тъй като основните концепции на Java са по-разбираеми когато се представят в конзолен режим.

2.1 Ключови думи и идентификатори в Java.

В езика Java са дефинирани 48 ключови думи. Тези думи заедно с използваните операции и синтактични правила дефинират езика Java. Ключовите думи не могат да бъдат използвани

като имена на променливи, методи или класове. По-голямата част от ключовите думи са заимствани от езика C++, като думите **const** и **goto** са запазени но не се използват. Ключовите думи в Java са представени в долната таблица.

Таблица 6.3.1

abstract	**boolean**	**break**	**byte**	**case**	**catch**
char	**class**	**const**	**continue**	**default**	**do**
double	**else**	**extends**	**final**	**finally**	**float**
for	**goto**	**if**	**implements**	**import**	**instanceof**
int	**interface**	**long**	**native**	**new**	**package**
private	**protected**	**public**	**return**	**short**	**static**
strictfp	**super**	**switch**	**synchronized**	**this**	**throw**
throws	**transient**	**try**	**void**	**volatile**	**while**

В Java, идентификаторите представляват имена, които се присвояват на променливи, функции (методи) или на някакви други елементи дефиниран в програмата. Идентификаторите могат да се състоят от един или няколко от допустимите за използване символи. Имената на променливите могат да започват с произволна буква от азбуката, знака за подчертаване или доларов знак. Следващите знаци могат да бъдат букви, числа, знак за подчертаване или доларов знак. Java е чувствителен към символния регистър, което значи, че малките и големите букви се различават. Например, идентификаторите **`myvar`** и **`MyVar`** са две различни наименования на променливи.

Ключовите думи не могат да се използват като идентификатори, а също така и имената на някои стандартни методи (функции), като **`println,`** например.

2.2 Основни типове данни

Java съдържа две основни категории от вградени типове данни: обектно-ориентирани и необектно-ориентирани. Обектно-ориентираните типове данни са дефинирани посредством класове.

Въпреки че Java е изцяло обектен език, той подържа някои основни типове данни, които могат да се използват в обектите. Като производен от C и C++, Java е близък до тях по отношение на множеството от типове данни които поддържа, но има и известни различия. Java подържа следните прости (необектни) типове данни: числови, булеви и символни данни.

- **Числовите типове данни** биват два вида – целочислени и реални типове данни. Целочислените типове данни са:

byte - 8 – битово цяло число (1 байт) (от **`-128`** до **`127`**)
short - 16 – битово цяло число (2 байта) (от **`-32768`** до **`32767`**)
int - 32 – битово цяло число (4 байта)
long - 64 – битово цяло число (8 байта)

8 – битовият тип данни **byte** заместват старият тип данни в C и C++, **char**. Java дава различна интерпретация за типа **char**, както ще бъде показано по-долу. В Java не съществува и определението *usigned* (беззнаков) за целочисления тип данни. Операторът за деклариране на променливи от определен тип данни има вида:

```
int ii,j,m1,n ;
```

Декларацията включва ключова дума за типа на данните (**`int`**) и наименования (идентификатори) на променливите - `ii,j,m1,n.`

- Реалните типове данни са:

float - 32 – битово реално число с плаваща точка
double- 64 – битово реално число с плаваща точка

Реалните типове данни и техните аритметични операции са реализирани според спецификацията определена в стандарта IEEE 754. За явно зададена реална числова стойност с плаваща точка, като 3.14, по подразбиране се използва типа данни **double**. Тя трябва явно да се преобразува в типа float ако се налага да се присвои на променлива от типа **float**.

- Символният тип данни в Java се различава от традиционната декларация на символни данни в езика C и C++. Тук типът данни char задава 16-битов символ в Unicode. Символите в Unicode са беззнакови 16 – битови стойности, които дефинират символни кодове в диапазона от 0 до 65535. Декларация от типа:

Char myChar = 'Q'

определя символна променлива, която се инициализира със стойността в Unicode на символа 'Q'. Като възприема като стандарт за своя символен тип данни множеството от символи в Unicode, езикът Java дава възможност приложенията написани с негова помощ да се интернационализират и локализират, с което се разширява изключително много пазара за тези приложения.

- **Булеви типове данни**. В Java има булев тип данни, който е от основните типове, като по този начин се утвърждава съществуващата в C и C++ практика разработчиците да дефинират запазени думи TRUE и FALSE, YES и NO и други конструкции, които представяха логически стойности (стойности на булеви променливи). В езика Java променливите от тип **boolean** приемат само стойности true и false – така типът данни **Boolean** става ясен и точен, за разлика от обичайната практика в C и C++. Типът **Boolean** не може да бъде преобразуван в числен тип данни.

В Java се използват данни, които се наричат ***литерали*** и представляват фиксирани стойности, представени във вид удобен за възприемане от потребителя. Например числото сто записано като 100 е литерал. Литералите се наричат и константи, но те са константи без наименование (неименувани константи). Този тип съответства на неименуваните константи използвани в език Pascal и C.

Литералите в Java могат да бъдат от всеки прост тип. Начинът, по който се представя даден литерал зависи от неговия тип. Символните константи, например се задават, като съответният символ се огради с единични кавички (апострофи) - **'a'**, **'%'** и т.н. Целочислените константи се записват като числа без десетична точка, а реалните константи - като числа с десетична точка. Например 10, -100 са целочислени константи (литерали), а 3.14159 - реална константа. По подразбиране, целочислените константи са от тип **int**. Ако трябва дадена константа да се разглежда като константа от тип **long**, е необходимо към числото да се добави символът **l** или **L**. Например **12** е **int**, а **12L** е **long** константа.

По подразбиране, литералите с плаваща точка са от тип **double**. Ако трябва да се зададе константа от тип **float**, към числото се добавя **f** или **F**. Например **12.43f** е константа от тип **float**.

Понякога в програмирането е по-удобно да се използва бройна система базирана на **8** или **16**, вместо десетична бройна система. Константите в шестнадесетична бройна система се записват, като преди шестнадесетичното число се запише '**0x**', а осмична константа - като пред осмичното число се запише '**0**'. Например **0xFF** е шестнадесетична константа, а **011** - осмична константа.

Java подържа и един друг тип литерал - низов литерал. Низът представлява набор от символи заградени с двойни кавички. Например, "**това е текстов низ**" е низова константа (литерал).