JAVA

Java je programovací jazyk a platforma. Java je vysokoúrovňový, robustný, objektovo orientovaný a bezpečný programovací jazyk. Program Java môže bežať na širokej škále počítačov, pretože nevykonáva pokyny priamo na počítači. Namiesto toho Java beží na Java Virtual Machine (JVM). Java je univerzálny programovací jazyk, ktorý sa používa vo všetkých odvetviach pre takmer všetky typy aplikácií. Akékoľvek hardvérové alebo softvérové prostredie, v ktorom beží program, sa nazýva platforma. Keďže Java má runtime prostredie (JRE) a API, nazýva sa platforma.

```
ClassSimple {
public static  void main(String args[]) {
System.out.println("HelloJava");
}
}
```

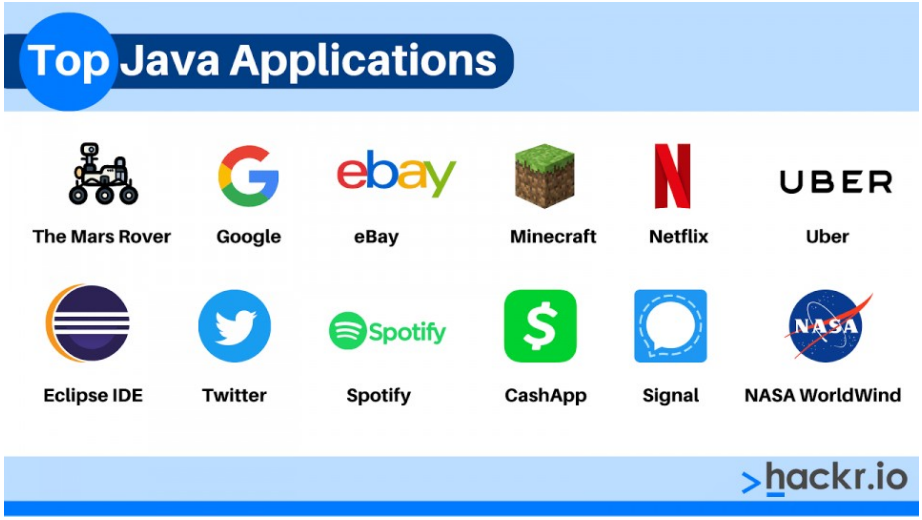
Typy Java aplikácií

1) Samostatná aplikácia - Samostatné aplikácie sú známe aj ako desktopové aplikácie alebo aplikácie v okne. Ide o tradičný softvér, ktorý musíme nainštalovať na každý stroj. Príkladmi samostatných aplikácií sú prehrávač médií, antivírus atď. AWT a Swing sa v jazyku Java používajú na vytváranie samostatných aplikácií.

2) Webová aplikácia - Aplikácia, ktorá beží na strane servera a vytvára dynamickú stránku, sa nazýva webová aplikácia. V súčasnosti sa na tvorbu webových aplikácií v Jave používajú technológie Servlet, JSP, Struts, Spring, Hibernate, JSF atď.

3) Podniková aplikácia - Aplikácia, ktorá je distribuovaná, ako napríklad bankové aplikácie atď., sa nazýva podniková aplikácia. Má výhody, ako je vysoká úroveň zabezpečenia, vyrovnávanie záťaže a klastrovanie. V jazyku Java sa EJB používa na vytváranie podnikových aplikácií.

4) Mobilná aplikácia - Aplikácia, ktorá je vytvorená pre mobilné zariadenia, sa nazýva mobilná aplikácia. V súčasnosti sa na tvorbu mobilných aplikácií používa Android a Java ME.



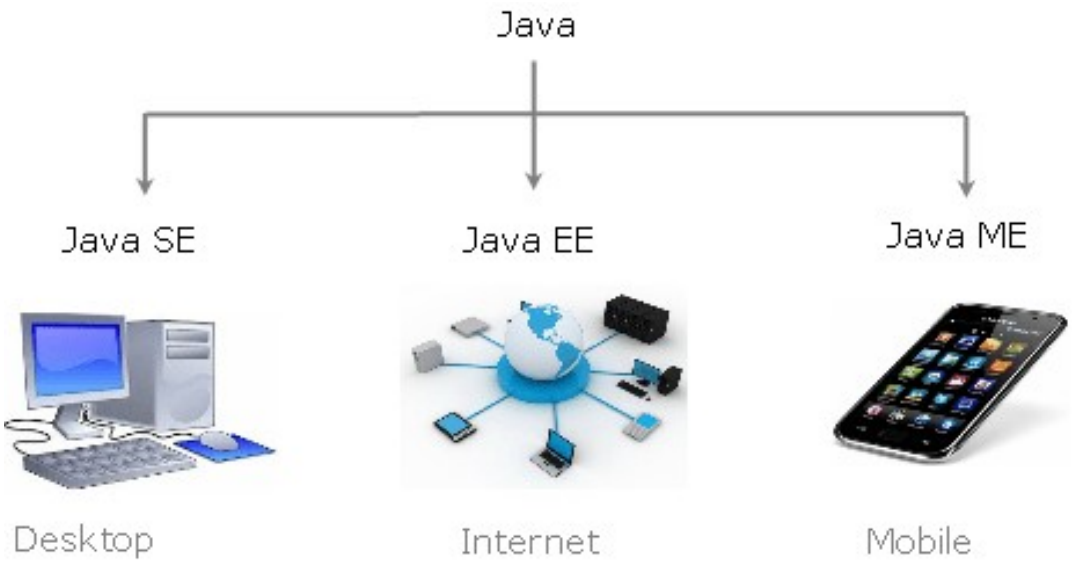
Platformy / edície Java

1) Java SE (Java Standard Edition) - Je to programovacia platforma Java. Zahŕňa programovacie rozhrania Java, ako sú java.lang, java.io, java.net, java.util, java.sql, java.math atď. Zahŕňa základné témy ako OOP, String, Regex, Výnimka, Vnútorné triedy, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection atď.

2) Java EE (Java Enterprise Edition) - Ide o podnikovú platformu, ktorá sa používa najmä na vývoj webových a podnikových aplikácií. Je postavený na platforme Java SE. Zahŕňa témy ako Servlet, JSP, Web Services, EJB, JPA atď.

3) Java ME (Java Micro Edition) - Ide o mikro platformu, ktorá sa venuje mobilným aplikáciám.

4) JavaFX - Používa sa na vývoj bohatých internetových aplikácií. Používa ľahké používateľské rozhranie API.



Vlastnosti Java

1. Jednoduché - Java sa veľmi ľahko učí a jej syntax je jednoduchá, čistá a ľahko pochopiteľná. Java odstránila mnoho komplikovaných a zriedkavo používaných funkcií, napríklad explicitné ukazovatele, preťaženie operátora atď. Nie je potrebné odstraňovať nereferencované objekty, pretože v jazyku Java existuje automatický zber odpadu.

2. Objektovo orientované - Java je objektovo orientovaný programovací jazyk. Všetko v Jave je objekt. Objektovo orientovaný znamená, že náš softvér organizujeme ako kombináciu rôznych typov objektov, ktoré zahŕňajú údaje aj správanie. Objektovo orientované programovanie (OOP) je metodika, ktorá zjednodušuje vývoj a údržbu softvéru tým, že poskytuje určité pravidlá. Základné koncepty OOP sú: Objekt,Trieda ,Dedičnosť ,Polymorfizmus,Abstrakcia,Zapuzdrenie.

3. Nezávislá od platformy - Java je nezávislá na platforme, pretože sa líši od iných jazykov, ktoré sú kompilované do strojov špecifických pre platformu, zatiaľ čo Java je jazyk určený na jednorázový zápis a spustenie kdekoľvek. Platforma je hardvérové alebo softvérov prostredie, v ktorom beží program. Existujú dva typy platforiem – softvérové a hardvérové. Java poskytuje softvérovú platformu.
Platforma Java sa líši od väčšiny ostatných platforiem v tom zmysle, že ide o softvérovú platformu, ktorá beží nad inými hardvérovými platformami. Má dve zložky: Runtime prostredie API (Aplikačné programové rozhranie) Java kód môže byť spustený na viacerých platformách, napríklad Windows, Linux, Sun Solaris, Mac/OS atď. Java kód je kompilovaný kompilátorom a konvertovaný na bajtkód. Tento bajtový kód je kód nezávislý od platformy, pretože ho možno spustiť na viacerých platformách, t. j. Write Once and Run Anywhere (WORA).

4. Zabezpečené - Java je známa najmä svojou bezpečnosťou. S Java môžeme vyvíjať systémy bez vírusov. Java je zabezpečená, pretože:
Žiadny explicitný ukazovateľ, Java programy bežia v karanténe virtuálneho stroja. Classloader: Classloader v jazyku Java je súčasťou prostredia Java Runtime Environment (JRE), ktoré sa používa na dynamické načítanie tried Java do virtuálneho stroja Java. Zvyšuje bezpečnosť tým, že oddeľuje balík pre triedy lokálneho súborového systému od tých, ktoré sú importované zo sieťových zdrojov.
Bytecode Verifier: Kontroluje fragmenty kódu na nelegálny kód, ktorý môže porušovať prístupové práva k objektom.
Security Manager: Určuje, ku ktorým prostriedkom môže trieda pristupovať, ako je napríklad čítanie a zápis na lokálny disk.
Jazyk Java štandardne poskytuje tieto cenné papiere. Určité zabezpečenie môže poskytnúť vývojár aplikácie aj výslovne prostredníctvom SSL, JAAS, kryptografie atď.

5.Robustný - Java je robustná, pretože: Používa silnú správu pamäte.
Java poskytuje automatický zber odpadu, ktorý beží na Java Virtual Machine, aby sa zbavil objektov, ktoré už Java aplikácia nepoužíva.
V jazyku Java existuje spracovanie výnimiek a mechanizmus kontroly typu. Všetky tieto body robia Javu robustnou.

6. Architektonicky neutrálny - Java je architektonicky neutrálna, pretože neexistujú žiadne funkcie závislé od implementácie, napríklad veľkosť primitívnych typov je pevná.

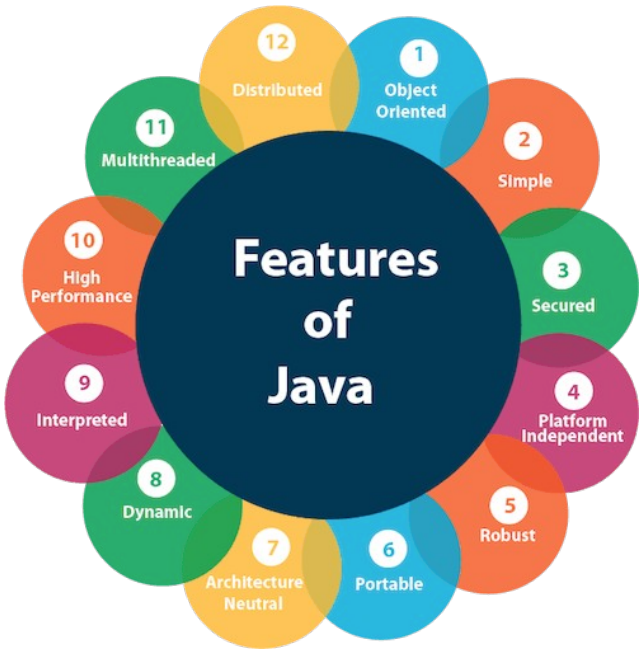
7. Prenosný - Java je prenosná, pretože vám uľahčuje prenos bajtového kódu Java na akúkoľvek platformu. Nevyžaduje žiadnu implementáciu.

8. Vysoký výkon - Java je rýchlejšia ako iné tradičné interpretované programovacie jazyky, pretože bajtový kód Java je „blízko“ natívnemu kódu.

9. Distribuovaný - Java je distribuovaná, pretože používateľom uľahčuje vytváranie distribuovaných aplikácií v jazyku Java. RMI a EJB sa používajú na vytváranie distribuovaných aplikácií. Táto funkcia Java nám umožňuje pristupovať k súborom volaním metód z akéhokoľvek počítača na internete.

10 .Viacvláknové - Vlákno je ako samostatný program, ktorý sa vykonáva súčasne. Môžeme písať Java programy, ktoré sa zaoberajú mnohými úlohami naraz definovaním viacerých vlákien. Hlavnou výhodou multi-threadingu je, že nezaberá pamäť pre každé vlákno. Zdieľa spoločnú pamäťovú oblasť. Vlákna sú dôležité pre multimédiá, webové aplikácie atď.

11. Dynamický - Java je dynamický jazyk. Podporuje dynamické načítanie tried. To znamená, že triedy sú načítané na požiadanie. Java podporuje dynamickú kompiláciu a automatickú správu pamäte .



Prvý program Java |

HELLO WORLD

```
class Simple{
public static void main(String args[]){
System.out.println("Hello World");
}
}
```

Pozrime sa, čo znamená class, public, static, void, main, String[], System.out.println() -

Slovo class sa používa na deklarovanie triedy v jazyku Java. Verejné kľúčové slovo je modifikátor prístupu, ktorý predstavuje viditeľnosť. To znamená, že je viditeľný pre všetkých.

static je slovo. Ak deklarujeme akúkoľvek metódu ako statickú, je známa ako statická metóda. Hlavnou výhodou statickej metódy je, že nie je potrebné vytvárať objekt na vyvolanie statickej metódy.

Metódu main() vykonáva JVM, takže na vyvolanie metódy main() nevyžaduje vytvorenie objektu. Takže šetrí pamäť.

void je návratový typ metódy. To znamená, že nevracia žiadnu hodnotu.

main predstavuje východiskový bod programu.

Argumenty typu String[] alebo Argumenty reťazca[] sa používajú ako argument príkazového riadka.

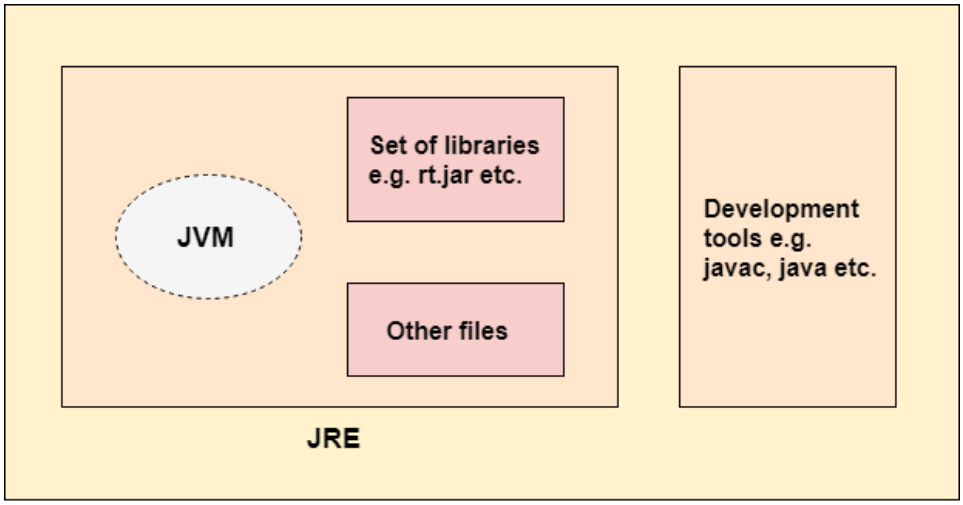
System.out.println() sa používa na tlač výpisu. Tu je System trieda, out je objekt triedy PrintStream, println() je metóda triedy PrintStream.

Rozdiel medzi JDK, JRE a JVM

1. JVM - JVM (Java Virtual Machine) je virtuálny stroj, pretože fyzicky neexistuje. Je to špecifikácia, ktorá poskytuje runtime prostredie, v ktorom je možné spustiť bajtový kód Java. Môže tiež spúšťať programy, ktoré sú napísané v iných jazykoch a skompilované do bajtového kódu Java. JVM sú dostupné pre mnohé hardvérové a softvérové platformy. JVM, JRE a JDK sú závislé na platforme, pretože konfigurácia každého OS sa navzájom líši. Java je však nezávislá na platforme. Existujú tri pojmy JVM: špecifikácia, implementácia a inštancia.
JVM vykonáva tieto hlavné úlohy: Načíta kód, Overí kód, Spustí kód, Poskytuje runtime prostredie.

2. JRE - JRE je skratka pre Java Runtime Environment. Java Runtime Environment je sada softvérových nástrojov, ktoré sa používajú na vývoj Java aplikácií. Používa sa na poskytovanie runtime prostredia. Ide o implementáciu JVM. Fyzicky existuje. Obsahuje sadu knižníc + ďalšie súbory, ktoré JVM používa za behu.

3.JDK - JDK je skratka pre Java Development Kit. Java Development Kit (JDK) je softvérové vývojové prostredie, ktoré sa používa na vývoj Java aplikácií a apletov. Fyzicky existuje. Obsahuje JRE + vývojové nástroje.
JDK obsahuje súkromný Java Virtual Machine (JVM) a niekoľko ďalších zdrojov, ako je interpret/loader (java), kompilátor (javac), archivátor (jar), generátor dokumentácie (Javadoc) atď. vývoj Java aplikácie.

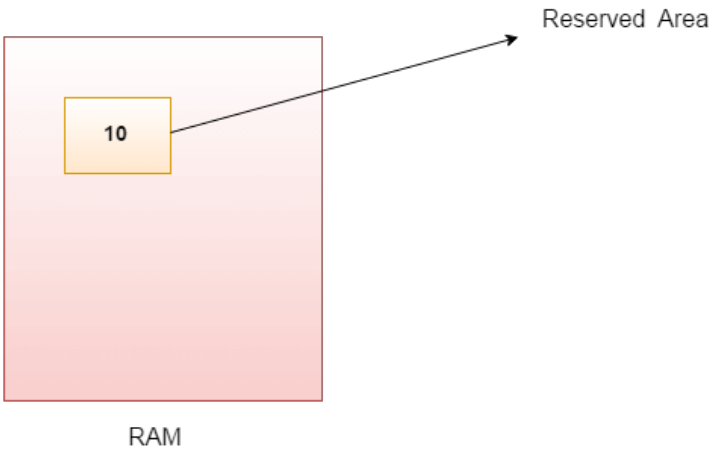


Premenné (Variables) Java

Premenná je kontajner, ktorý obsahuje hodnotu počas vykonávania programu Java. Premennej je priradený dátový typ. Premenná je názov miesta v pamäti. V jave existujú tri typy premenných: lokálne, inštancie a statické. V Jave existujú dva typy dátových typov: primitívne a neprimitívne.

Variable : premenná je názov vyhradenej oblasti alokovanej v pamäti. Inými slovami, je to názov pamäťového miesta. Je to kombinácia "vary + schopný", čo znamená, že jej hodnotu možno zmeniť.

int data=50; //Here data is variable.



Typy premenných (variables) :

1. local variable (lokálna premenná)
2. instance variable (inštančná premenná)
3. static variable (statická premenná)

1. Local Variable (Lokálna Premenná) : premenná deklarovaná v tele metódy sa nazýva lokálna premenná. Túto premennú môžete použiť iba v rámci tejto metódy a ostatné metódy v triede ani nevedia, že premenná existuje. Lokálna premenná nemôže byť definovaná pomocou kľúčového slova "static".

```
public class A {
int n=90; //local variable.
}
```

2. Instance Variable (Premenná Inštancie) : Premenná deklarovaná vo vnútri triedy, ale mimo tela metódy, sa nazýva premenná inštancie. Nie je deklarovaná ako statická. Nazýva sa premenná inštancie, pretože jej hodnota je špecifická pre inštanciu a nie je zdieľaná medzi inštanciami.

```
public class A {
public static void main(String args[])
{
int data=50; //instance variable.
}
}
```

3. Static variable (Statická Premenná) : .premenná, ktorá je deklarovaná ako statická, sa nazýva statická premenná. Nemôže byť lokálna. Môžete vytvoriť jednu kópiu statickej premennej a zdieľať ju medzi všetkými inštanciami triedy. Pridelenie pamäte pre statické premenné sa uskutoční iba raz, keď sa trieda načíta do pamäte.

Príklad premennej Java Pridajte dve čísla :

```
public class Simple {
public static void main(String[] args) {
int a=10;
int b=10;
int c=a+b;
System.out.println(c);
}
}
```

OUTPUT : 20

Príklad premennej Java Rozšírenie :

```
public class Simple {
public static void main(String[] args) {
int a=10;
float f=a;
System.out.println(a);
System.out.println(f);
}
}
```

OUTPUT : 10

10.0

Typy údajov (data types) v jazyku Java

Dátové typy určujú rôzne veľkosti a hodnoty, ktoré môžu byť uložené v premennej. V Jave existujú dva typy dátových typov :

1. Primitívne typy údajov - medzi primitívne typy údajov patria boolean, char, byte, short, int, long, float a double.

2. Neprimitívne typy údajov - medzi neprimitívne typy údajov patria triedy, rozhrania a polia.

Primitívne typy údajov Java : v jazyku Java sú primitívne dátové typy stavebnými kameňmi manipulácie s dátami. Toto sú najzákladnejšie dátové typy dostupné v jazyku Java:

1. Typ údajov Boolean - typ údajov Boolean sa používa na uloženie iba dvoch možných hodnôt: true a false. Tento typ údajov sa používa pre jednoduché príznaky, ktoré sledujú pravdivé/nepravdivé podmienky.
Dátový typ Boolean špecifikuje jeden bit informácie, ale jeho „veľkosť“ sa nedá presne definovať.

boolean one = false

2. Typ údajov Byte (bit) - typ údajov Byte je príkladom primitívneho typu údajov. Jeho hodnota sa pohybuje v rozmedzí -128 až 127. Jeho minimálna hodnota je -128 a maximálna hodnota je 127. Predvolená hodnota je 0. Dátový typ byte sa používa na šetrenie pamäte vo veľkých poliach, kde je úspora pamäte najviac potrebná. Šetrí miesto, pretože bajt je 4-krát menší ako celé číslo. Môže sa použiť aj namiesto dátového typu „int“.

byte a = 10, byte b = -20

3. Typ údajov Short (krátky) - jeho hodnota sa pohybuje medzi -32 768 až 32 767 . Jeho minimálna hodnota je -32 768 a maximálna hodnota je 32 767. Jeho predvolená hodnota je 0. Krátky dátový typ možno použiť aj na šetrenie pamäte, rovnako ako bajtový dátový typ. Krátky dátový typ je 2-krát menší ako celé číslo.

short s = 10000, short r = -5000

4. Typ údajov Int - jeho rozsah hodnôt leží medzi - 2 147 483 648 až 2 147 483 647 . Jeho minimálna hodnota je - 2 147 483 648 a maximálna hodnota je 2 147 483 647. Jeho predvolená hodnota je 0. Dátový typ int sa vo všeobecnosti používa ako predvolený dátový typ pre integrálne hodnoty, pokiaľ nie je problém s pamäťou.

int a = 100000, int b = -200000

5. Typ údajov Long (dlhý) - jeho rozsah hodnôt leží medzi -9 223 372 036 854 775 808 až 9 223 372 036 854 775 807 . Jeho minimálna hodnota je - 9,223,372,036,854,775,808 a maximálna hodnota je 9,223,372,036,854,775,807. Jeho predvolená hodnota je 0. Typ údajov long sa používa, keď potrebujete rozsah hodnôt väčší ako tie, ktoré poskytuje int.

long a = 100000L, long b = -200000L

6. Typ údajov Float (plavajuci) - typ údajov float s jednoduchou presnosťou a pohyblivou desatinnou čiarkou. Rozsah jeho hodnôt je neobmedzený. Odporúča sa použiť float (namiesto double), ak potrebujete ušetriť pamäť vo veľkých poliach čísel s pohyblivou desatinnou čiarkou. Typ údajov float by sa nikdy nemal používať pre presné hodnoty, ako je mena. Jeho predvolená hodnota je 0,0F.

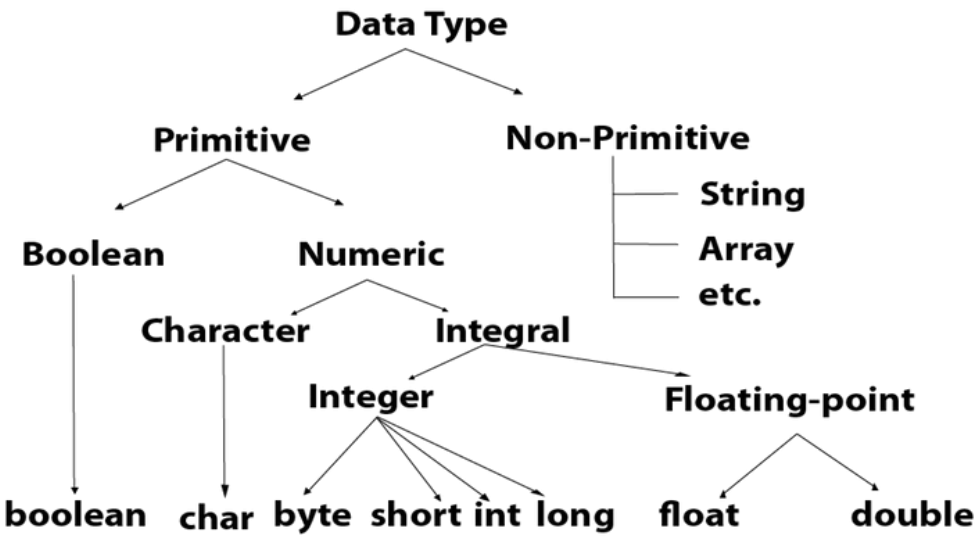
float f1 = 234.5f

7. Typ údajov Double (dvojitý) - typ údajov double s pohyblivou desatinnou čiarkou s dvojitou presnosťou. Rozsah jeho hodnôt je neobmedzený. Dátový typ double sa vo všeobecnosti používa pre desiatkové hodnoty, rovnako ako float. Typ údajov double by sa tiež nikdy nemal používať pre presné hodnoty, ako je napríklad mena. Jeho predvolená hodnota je 0,0 d.

double d1= 12.3

8. Typ údajov Char (znak) - jeho rozsah hodnôt leží medzi '\u0000' alebo 0 až '\uffff' alebo 65 535 . Typ údajov char sa používa na ukladanie znakov.

char letterA = 'A'



Operátori v jazyku Java

Operátor v jazyku Java je symbol, ktorý sa používa na vykonávanie operácií. Napríklad: +, -, *, / atď.
V jazyku Java existuje veľa typov operátorov :

1. Jednotný operátor

2. Aritmetický operátor

3.Operátor zmeny

4.Relačný operátor

5.Bitový operátor

6.Logický operátor

7.Ternárny operátor

8. Operátor pridelenia

Operators	Precedence
postfix	*expr++ expr--*
unary	*++expr --expr +expr -expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= *instanceof*
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

1. Operátor Java Unary - unárne operátory Java vyžadujú iba jeden operand. Unárne operátory sa používajú na vykonávanie rôznych operácií, napr. zvýšenie/zníženie hodnoty o jednu, negovanie výrazu, invertovanie hodnoty boolean.

Príklad unárneho operátora Java: ++ a – :

```
public class OperatorExample {
public static void main(String args[]) {
int x=10;
System.out.println(x++);  //10(11)
System.out.println(++x);  //12
System.out.println(x--);  //12(11)
System.out.println(--x);  //10
}
}
```

OUTPUT : 10

12

12

10

Príklad 2 unárneho operátora Java: ++ a – :

```
public class OperatorExample {
public static void main(String args[]) {
int a=10;
int b=10;
System.out.println(a++ + ++a);  //10+12=22
System.out.println(b++ + b++);  //10+11=21

}
}
```

OUTPUT : 22

21

Príklad unárneho operátora Java: ~ a ! :

```
public class OperatorExample {
public static void main(String args[]) {
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a);   //-11(minus of total positive value which starts from 0).
System.out.println(~b);  //9 (positive of total minus, positive starts from 0).
System.out.println(!c);  //false (opposite of boolean value).
System.out.println(!d);  //true.
}
}
```

OUTPUT : -11

9

false

true

2. Aritmetické operátory Java - aritmetické operátory Java sa používajú na vykonávanie sčítania, odčítania, násobenia a delenia. Fungujú ako základné matematické operácie.

Príklad aritmetického operátora Java :

```
public class OperatorExample {
public static void main(String args[]) {
int a=10;
int b=5;
System.out.println(a+b);  //15
System.out.println(a-b);  //5
System.out.println(a*b);  //50
System.out.println(a/b);  //2
System.out.println(a%b);  //0
}
}
```

OUTPUT : 15

5

50

2

0

Príklad aritmetického operátora Java - Výraz :

```
public class OperatorExample {
public static void main(String args[]) {
System.out.println(10*10/5+3-1*4/2);
}
}
```

OUTPUT : 21

3. Operátor ľavého posunu Java - Java operátor posunu doľava << sa používa na posunutie všetkých bitov v hodnote doľava v zadanom počte opakovaní.

Príklad operátora Java Shift :

```
public  classOperatorExample {
public static void main(String args[]) {
System.out.println(10<<2);  //10*2^2=10*4=40
System.out.println(10<<3);  //10*2^3=10*8=80
System.out.println(20<<2);  //20*2^2=20*4=80
System.out.println(15<<4);  //15*2^4=15*16=240
}
}
```

OUTPUT : 40
80
80
240

4. Operátor pravého posunu Java - operátor posunu vpravo >> Java sa používa na posunutie hodnoty ľavého operandu doprava o počet bitov špecifikovaný pravým operandom.

Príklad operátora Java Right Shift :

```
public OperatorExample {
public static void main(String args[]) {
System.out.println(10>>2);  //10/2^2=10/4=2
System.out.println(20>>2);  //20/2^2=20/4=5
System.out.println(20>>3);  //20/2^3=20/8=2
}
}
```

OUTPUT : 2

5

2

5. Príklad operátora Java AND: Logical and Bitwise - logický operátor nekontroluje druhú podmienku, ak je prvá podmienka nepravdivá. Kontroluje druhú podmienku iba vtedy, ak je prvá pravdivá.

Operátor bitový vždy kontroluje obe podmienky, či je prvá podmienka pravdivá alebo nepravdivá.

```
public class OperatorExample {
public static void main(String args[]) {
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);  //false  true =false
System.out.println(a<b&a<c);  //false  true = false
}
}
```

OUTPUT : false

false

Príklad operátora Java AND: Logical && vs Bitwise & :

```
public class OperatorExample {
public static void main(String args[]) {
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a++<c);  //false && true = false
System.out.println(a);  //10because second condition is not checked
System.out.println(a<b&a++<c);   //false && true = false
System.out.println(a);  //11 because second condition is checked
}
}
```

OUTPUT : false
10
false
11

6. Príklad operátora Java OR: Logický || a Bitwise | - Logické || operátor nekontroluje druhú podmienku, ak je prvá podmienka pravdivá. Kontroluje druhú podmienku iba vtedy, ak je prvá nepravdivá.

Bitový | operátor vždy skontroluje obe podmienky, či je prvá podmienka pravdivá alebo nepravdivá.

```
public class OperatorExample {
public static void main(String args[]) {
int a=10;
int b=5;
int  c=20;
System.out.println(a>b||a<c);  //true || true = true
System.out.println(a>b|a<c);   //true | true = true

System.out.println(a>b||a++<c);  //true  || true = true
System.out.println(a);  //10 because second condition is not checked
System.out.println(a>b|a++<c);  //true | true  = true
System.out.println(a);  //11 because second condition is checked
}
}
```

OUTPUT : true
true
true
10
true
11

7. Ternárny operátor Java - operátor Java Ternary sa používa ako náhrada jedného riadku za príkaz if-then-else a často sa používa v programovaní v jazyku Java. Je to jediný podmienený operátor, ktorý má tri operandy.

Príklad ternárneho operátora Java :

```
public class OperatorExample {
public static void main(String  args[]) {
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}
}
```

OUTPUT : 2

```
public class OperatorExample {
public static void main(String args[]) {
int a=10;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}
}
```

OUTPUT : 5

8. Operátor priradenia Java - operátor priradenia Java je jedným z najbežnejších operátorov. Používa sa na priradenie hodnoty napravo k operandu naľavo.

Príklad operátora priradenia Java :

```
public class OperatorExample {
public static void main(String args[]) {
int a=10;
int b=20;
a+=4;  //a=a+4 (a=10+4)
b-=4;  //b=b-4  (b=20-4)
System.out.println(a);
System.out.println(b);
}
}
```

OUTPUT : 14

16

```
public class OperatorExample {
public static void main(String[] args) {
int a= 10;
a+=3;  //10+3
System.out.println(a);
a-=4;  //13-4
System.out.println(a);
a*=2; //9*2
System.out.println(a);
a/=2;  //18/2
System.out.println(a);
}
}
```

OUTPUT : 13
9
18
9

Príklad operátora priradenia Java: Pridanie SHORT :

```
public class OperatorExample {
public static void main(String args[]) {
short a=10;
short b=10;
a=(short)(a+b);  //20 which is int  now converted to short
System.out.println(a);
}
}
```

OUTPUT : 20

Kontrolné vyhlásenia Java | Riadenie toku v jazyku Java

(Control Statements and Control Flow)

Java kompilátor vykoná kód zhora nadol. Príkazy v kóde sa vykonávajú podľa poradia, v akom sa objavujú. Java však poskytuje príkazy, ktoré možno použiť na riadenie toku kódu Java. Takéto príkazy sa nazývajú príkazy riadiaceho toku. Je to jedna zo základných vlastností Java, ktorá poskytuje plynulý tok programu.

Java poskytuje tri typy príkazov toku riadenia -

1.Decision making statement (rozhodovacie vyhlásenia) -

1. if statement.

2. switch statement.

2. Loop statement (výkaz slučky) -

1.do while loop (urobiť while slučku).

2. while loop (pričom slučka).

3. for loop (pre slučku).

4. for-each loop (pre každu slučku).

3. Jump statements (skokové vyhlásenia) -

1. break statement (vyhlásenie o prerušení).

2. continue statement (pokračovat vo vyhlásení).

1. Decision-Making statements (rozhodovacie vyhlásenie) :

rozhodovacie výroky rozhodujú o tom, ktorý výrok a kedy vykonať. Rozhodovacie príkazy vyhodnocujú booleovský výraz a riadia tok programu v závislosti od výsledku poskytnutej podmienky. V jazyku Java existujú dva typy rozhodovacích príkazov, príkaz If a príkaz switch.

1. If Statement : v jazyku Java sa na vyhodnotenie podmienky používa príkaz if. Riadenie programu je odklonené v závislosti od konkrétnych podmienok. Podmienka príkazu If dáva boolovskú hodnotu, buď true alebo false. V jazyku Java sú nižšie uvedené štyri typy príkazov if :

Simple if statement :

1. if-else statement
2. if-else-if ladder
3. Nested if-statement

1) Jednoduchý príkaz if (simple if statement) : je to najzákladnejší príkaz spomedzi všetkých príkazov toku riadenia v Jave. Vyhodnocuje boolovský výraz a umožňuje programu zadať blok kódu, ak sa výraz vyhodnotí ako true.

```
public class  Student {
public static void  main(String[args) {
int x = 10;
int y = 12;
if(x+y > 20) {
System.out.println("x + y is greater than 20");
}
}
}
```

OUTPUT : x+y is greater than 20

2) if-else statement : príkaz if-else je rozšírením príkazu if, ktorý používa ďalší blok kódu, blok else. Blok else sa vykoná, ak je podmienka bloku if vyhodnotená ako nepravdivá.

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y < 10) {
System.out.println("x + y is less than10");
}else {
System.out.println("x + y is greater than 20");
}
}
}
```

OUTPUT : x + y is greater than 20

3) if-else-if ladder : príkaz if-else-if obsahuje príkaz if, za ktorým nasledujú viaceré príkazy else-if. Inými slovami, môžeme povedať, že je to reťaz príkazov if-else, ktoré vytvárajú rozhodovací strom, do ktorého môže program vstúpiť do bloku kódu, kde je podmienka pravdivá. Na konci reťazca môžeme definovať aj príkaz else.

```
public classStudent {
public static void main(String[]args) {
String city = "Delhi";
if(city == "Meerut") {
System.out.println("city  is meerut");
}else if (city == "Noida") {
System.out.println("city is noida");
}else if(city == "Agra") {
System.out.println("city is agra");
}else {
System.out.println(city);
}
}
}
```

OUTPUT : Delhi

4) Nested if-statement : vo vnorených príkazoch if môže príkaz if obsahovať príkaz if alebo if-else v inom príkaze if alebo else-if.

```
public  class Student {
public static void main(String[]args){
String address ="Delhi, India";

if(address.endsWith("India")){
if(address.contains("Meerut")) {
System.out.println("Your city is Meerut");
}else if(address.contains("Noida")) {
System.out.println("Your city is Noida"){
}else{
System.out.println(address.split(",")[0]);
}
}else {
System.out.println("You are not living in India");
}
}
}
```

OUTPUT : Delhi

2. Switch Statement : v jazyku Java sú príkazy Switch podobné príkazom if-else-if. Príkaz switch obsahuje viacero blokov kódu nazývaných prípady a jeden prípad (case) sa vykoná na základe premennej, ktorá sa prepína. Namiesto príkazov if-else-if je jednoduchšie použiť príkaz switch. To tiež zvyšuje čitateľnosť programu.Pri používaní príkazov switch si musíme všimnúť, že výraz case bude rovnakého typu ako premenná. Bude to však aj konštantná hodnota. Switch umožňuje použiť iba premenné typu int, string a Enum.

Body ktoré treba poznamenať o príkaze switch :

1. Premenné veľkosti písmen môžu byť int, short, byte, char alebo enumeration.

2. Prípady nemožno duplikovať.

3. Predvolený príkaz sa vykoná, keď ktorýkoľvek prípad nezodpovedá hodnote výrazu. Je to voliteľné.

4. Príkaz Break ukončí prepínací blok, keď je podmienka splnená.

6. Je voliteľný, ak sa nepoužije, vykoná sa ďalší prípad.

7. Pri používaní príkazov switch si musíme všimnúť, že výraz case bude rovnakého typu ako premenná. Bude to však aj konštantná hodnota.

```
public class Student implements Cloneable {
public static void main(String[] args) {
int num = 2;
switch (num){
case0:
System.out.println("number is 0");
break;
case 1:
System.out.println("number is 1");
break;
default:
System.out.println(num);
}
}
}
```

OUTPUT : 2

3. Loop Statements : pri programovaní niekedy potrebujeme vykonať blok kódu opakovane, kým sa niektorá podmienka vyhodnotí ako pravdivá. Príkazy cyklu sa však používajú na vykonanie množiny pokynov v opakovanom poradí. Vykonanie súboru inštrukcií závisí od konkrétnej podmienky. V Jave máme tri typy cyklov, ktoré sa vykonávajú podobne. Existujú však rozdiely v ich syntaxi a čase kontroly stavu.

1.for loop

2 while loop

3. do-while loop

1) Java for loop : umožňuje nám inicializovať premennú cyklu, skontrolovať stav a inkrementovať/znižovať v jednom riadku kódu. Cyklus for používame iba vtedy, keď presne vieme, koľkokrát chceme vykonať blok kódu.

```
public class Calculattion {
public static void main(String[] args) {
// TODO Auto-generated method stub
int sum = 0;
for(int j = 1; j<=10; j++) {
sum = sum + j;
}
System.out.println("The sum of first 10 natural numbers is " + sum);
}
}
```

OUTPUT : The sum of first 10 natural numbers is 55

2) Java for-each loop : Java poskytuje vylepšenú slučku for na prechádzanie dátovými štruktúrami, ako je pole alebo kolekcia. V slučke for-each nemusíme aktualizovať premennú slučky.

```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
String[] names = {"Java","C","C++","Python","JavaScript"};
System.out.println("Printing the content of the array names:\n");
for (String name:names) {
System.out.println(name);
}
}
}
```

OUTPUT : Printing the content of the array names :

Java

C

C++

Python

Javascript

3) Java while loop : cyklus while sa tiež používa na opakované opakovanie počtu príkazov. Ak však vopred nepoznáme počet iterácií, odporúča sa použiť cyklus while. Na rozdiel od cyklu for sa inicializácia a inkrementácia/zníženie neuskutočňuje vo vnútri príkazu loop v slučke while. Je tiež známy ako slučka riadená vstupom, pretože podmienka sa kontroluje na začiatku slučky. Ak je podmienka pravdivá, telo slučky sa vykoná; inak sa vykonajú príkazy po slučke.

```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
int i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
while(i<=10) {
System.out.println(i);
i = i + 2;
}
}
}
```

OUTPUT : Printing the list of first 10 even numbers

0

2

4

6

8

10

4) Java do-while loop : cyklus do-while kontroluje stav na konci cyklu po vykonaní príkazov cyklu. Keď nie je známy počet iterácií a musíme vykonať cyklus aspoň raz, môžeme použiť cyklus do-while. Je tiež známy ako výstupná riadená slučka, pretože podmienka nie je vopred kontrolovaná.

```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
int  i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
do {
System.out.println(i);
i = i + 2;
}while (i<=10);
}
}
```

OUTPUT : Printing the list of first 10 even numbers

0

2

4

6

8

10

3. Jump Statements : príkazy jump sa používajú na prenos ovládania programu na konkrétne príkazy. Inými slovami, skokové príkazy prenesú riadenie vykonávania do inej časti programu. V jazyku Java existujú dva typy príkazov jumpu, break a continue.

1) Java break statement : príkaz break sa používa na prerušenie aktuálneho toku programu a prenos riadenia na nasledujúci príkaz mimo príkazu slučky alebo switchu. V prípade vnorenej slučky však preruší iba vnútornú slučku. Príkaz break nemôže byť použitý samostatne v programe Java, t.j. môže byť napísaný iba vo vnútri príkazu slučky alebo switchu. Príklad príkazu break so slučkou for .

```
public  class BreakExample {

public static void main(String[] args) {
// TODO Auto-generated method stub
for(int i = 0;i<= 10; i++) {
System.out.println(i);
if(i==6) {
break;
}
}
}
}
```

OUTPUT : 0
1
2
3
4
5
6

```
public class Calculation {

public static void main(String[] args) {
// TODO Auto-generated method stub
a:
for(int  i = 0; i<= 10; i++) {
b:
for(int j = 0; j<=15;j++){
c:


for (int k= 0; k<=20; k++) {
System.out.println(k);
if(k==5) {
break a;
}
}

}
}
}
```

OUTPUT :

2) Java continue statement : na rozdiel od príkazu break príkaz continue nepreruší cyklus, zatiaľ čo preskočí špecifickú časť cyklu a okamžite preskočí na ďalšiu iteráciu cyklu.

```
public class ContinueExample {

public static void main(String[] args) {
// TODO Auto-generated method stub

for(int i = 0; i<= 2; i++) {

for (int j = i; j<=5; j++) {

if(j == 4) {
continue;
}
System.out.println(j);
}
}
}

}
```

OUTPUT :

Java If-else Statement

(If-else vyhlásenie)

Na testovanie podmienky sa používa príkaz Java if. Kontroluje boolovskú podmienku: true alebo false. V jazyku Java existujú rôzne typy príkazov if :

1) **if statement**
2) **if-else statement**
3) **if-else-if ladder**
4) **nested if statement**

1) Java if Statement : příkaz if v jazyku Java testuje podmienku. Ak je podmienka pravdivá, vykoná sa blok if.

```
// Java Program to demonstate the use of if statement.
public class IfExample {
public static void main(String[] args) {
//defining an 'age' variable
int age=20;
//checking the age
if(age>18){
System.out.print("Age is greater than 18");
}
}
}
```

OUTPUT : Age is greater than 18

2) Java if-else Statement : příkaz Java if-else tiež testuje podmienku. Ak je podmienka pravdivá, vykoná sa blok if, inak sa vykoná blok.

```
// A Java Program to demonstrate the use of if-else statement.
//  It is a program of odd and even number.
public class IfElseExample {
public static void main(String[] args) {
// defining a variable.
int number=13;
//Check if the number is divisible by 2 or not
if(number%2==0) {
System.out.println("even number");
} else {
System.out.println(odd number");
}
}
}
```

OUTPUT : odd number

Používanie ternárneho operátora (ternary operator) : na vykonanie úlohy príkazu if...else môžeme použiť aj ternárny operátor (? :). Je to skrátený spôsob kontroly stavu. Ak je podmienka pravdivá, výsledok ? sa vráti. Ak je však podmienka nepravdivá, vráti sa výsledok :.

```
public class IfElseTernaryExample {
public static void main(String[] args) {
int number=13;
// Using ternary operator.
String output=(number%2==0)?"even number":"odd number";
System.out.println(output);
}
}
```

OUTPUT : odd number

3) Java if-else-if ladder Statement : príkaz if-else-if ladder vykoná jednu podmienku z viacerých príkazov.

```
// Java Program to demonstrate the use of If else-if ladder.
// It is a program of grading system for fail, D grade, C grade, Bgrade, A.
grade and A+.
public class IfElseIfExample {
public static void main(String[] args) {
int marks=65;
if(marks<50) {
System.out.println("fail");
}
else if(marks>=50 && marks<60) {
System.out.println("D grade");
}
else if(marks>=60 && marks<70){
System.out.println("C grade");
}
else if(marks>=70 && marks<80) {
System.out.println("B grade");
}
else if(marks>=80 && marks<90) {
System.out.println("A grade");
} else if(marks>=90 && marks<100) {
System.out.println("A+ grade");
}else{
System.out.println("Invalid!");
```

```
}
}
}
```

OUTPUT : C grade

4) Java Nested if statement : vnorený príkaz if predstavuje blok if v inom bloku if. Tu sa vnútorná podmienka bloku if vykoná iba vtedy, keď je podmienka vonkajšieho bloku pravdivá.

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample {
public static void main(String[] args) {
// Creating two variables for age and weight.
int age= 20;
int weight= 80;
// applying condition on age and weight.
if(age>=18){
if(weight>50){
System.out.println("You are eligible to donate blood");
}
}
}
}
```

OUTPUT : You are eligible to donate blood

```
// Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample2 {
public static void main(String[] args) {
// Creating two variables for age and weight.
int age=25;
int weight= 48;
//  applying condition on age and weight.
if(age>=18) {
if(weight>50) {
System.out.println("You are eligible todonate blood");
} else {
System.out.println("You are not eligible to donate blood");
}
} else {
System.out.println("Age must begreater than 18");
}
}
}
```

OUTPUT : You are not eligable to donate blood