**Java nədir**

Java komputerlər, mobil cihazlar, oyun konsolları, tibbi cihazlar və digərləri də daxil olmaqla milyardlarla cihazda işləyən, geniş istifadə olunan obyekt yönümlü proqramlaşdırma dili və proqram plartformasıdır. Java’nın qaydaları və sintaksisi C və C++ dillərinə əsaslanır.Java 1991-ci ildə Sun Microsystems şirkətindən James Gosling tərəfindən icad olunduqda başlıca məqsəd “write once, run everywhere” (bir dəfə yaz, hər yerdə işlət) idi.

**Java’nın əsas elementləri**

* Data types(Verilənlərin tipləri)
* Variables(Dəyişənlər & Verilənlər)
* Operators(Operatorlar)
* String class(String class’lar)
* Control Statements |Control flow(Idarə etmə şərtləri)
* Loops (Dövrlər)
* Arrays (Massivlər)

Nəyə görə bunlara Java’nın əsas elementləri deyilir? Çünki bu əsas elementlər sayəsində biz basic proqramlar yaza bilərik.Yuxarı səviyyə elementlər bu əsas elementlərin üzərində qurulub. Bu elementlər olmasa proqramlaşdırma da olmaz.

**Data types (Verilənlərin Tipləri)**

Java’da Verilənlərin tipləri 2 yerə bölünür, Primitve və Reference .

**Primitive type**

Primitive type dəyişənlər stack yaddaşda ardıcıl olaraq yerləşir. Bu type’lar kiçik hərflərlə yazılır.

Primitive type’ların istifadə məqsədi, yalnız dəyişənin qiymətini özündə saxlamaqdır.

Primitive type’lar 3 yerə bölünür.

Logical type(Məntiqi) – boolean

Numerical type (Ədədi) – byte, short, int, long, float, double

Character type (Simvol). – char

Logical type variable(dəyişən) yalnız true və ya false dəyər ala bilərlər.

Numerical types variable’ların default(susmaya görə) dəyərləri 0 olur.Numerical type da özü 2 yerə bölünür Integer types və Floating-Point types.

Integer types – byte ,short ,int ,long

Bu typeların hamısı signed, positive və negative’dir. Java’da only-positive və unsigned integers type’lar dəstəklənmir

Floating- Porint types – float , double

Float type(primitive) variable təyin edərkən dəyişənin yanına f yazılmalıdır. Əks təqdirdə avtomatik olaraq double başa düşüləcək və compile error verəcək.

Character type variable’lar daxillərində yalnız tək dırnaq arasında 1 simvol saxlaya bilər.

**Data type ranges (Verilənlərin tiplərinin aralıqları)**

byte (Yaddaşda tutduğu yer 1 byte) -128 to 127

short (Yaddaşda tutduğu yer 2 bytes) -32,768 to 32,767

int (Yaddaşda tutduğu yer 4 bytes) -2,147,483,648 to 2,147,483,647

long (Yaddaşda tutduğu yer 8 bytes) -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

float(Yaddaşda tutduğu yer 4 bytes) vergüldən sonra 6 və ya 7 rəqəm üçün nəzərdə tutulub

double(Yaddaşda tutduğu yer 8 bytes) vergüldən sonra 16 rəqəm üçün nəzərdə tutulub

boolean(Yaddaşda tutduğu yer 1 byte) true və ya false

char (Yaddaşda tutduğu yer 2 bytes) Simvollar və ya onlara uyğun olaraq ASCII kodları

**Reference type**

Reference type’a həm də Object type və ya Wrapper type da deyilir. Object type deyilməsinin səbəbi Reference type dəyişənlərin method’larının olmasıdır.

Reference type dəyişənlər Heap yaddaşda oyuqlar şəklində saxlanılır.

Reference type variable’ların default(susmaya görə) dəyəri null’dır. Çünki Reference type dəyişənlər object olurlar.

**Variables(Dəyişənlər)**

Dəyişən adı təyin edərkən biz A-Z, a-z, \_ , $ ,0-9, unicode kimi simvolların hər birini istifadə edə bilərik.

Dəyişən adı əsasən 4 tipdən istifadə olunaraq təyin olunur camelCase, PascalCase, snake\_case, kebab-case.Javada isə əsasən camelCase və snake\_case daha çox istifadə edilir.

**Operators(Operatorlar)**

1.Arithmetic Operators(Hesab operatorları)

2.Relational Operators(Əlaqəli operatorlar)

3.Logical Operators(Məntiqi operatorlar)

4.Assignment Operators(Əlaqəli işləyən operatorlar)

5.Misc Operators(Müxtəlif operatorlar)

**Arithmetic operators**(Hesab operatorları)

/ - div bölmə adlanır , tam hissə götürülür

% - mod bölmə adlanır , qalıq hissə götürülür

Increment - num++

Declerement - num—

num++ && num-- gedən proses özündən sonraki sətrə aid olur.

++num && --num həmin sətr daxilində artıq proses getmiş olur

int num = 10;

System.out.println(num++); yazarsaq ekrana **10** çıxacaq lakin artıq yaddaşda num **11** olaraq qalacaq və daha sonra num dəyişənini ekrana verərsək **11** kimi çıxacaq

**Relational Operators**(Əlaqəli operatorlar)

Müqayisə operatorları bizə true və ya false dəyər qaytarır

== 🡪 bərabərdirmi?

!= 🡪bərabərliyin əksi, bərabər deyilmi? yəni not equals

> 🡪 böyükdürmü?

< 🡪 kiçikdirmi?

>= 🡪böyük bərabərdirmi?

<= 🡪kiçik bərabərdirmi?

Bu operatorlar bizə boolean dəyər qayatırlar. Və əsasən if şərti daxilində və ya loop’larda istifadə olunurlar.

**Logical Operators**(Məntiqi operatorlar)

&& 🡪and(və) deməkdir. Verilən şərtləri birləşdirir və onların hər biri doğru olduğu halda bizə true dəyər qayıdır.

|| 🡪 or(və ya) deməkdir. Verilən şərtləri birləşdirir və onlardan heç olmasa biri doğru olduğu halda bizə true dəyər qayıdır.

! 🡪 not(inkar) deməkdir. Verilən şərtin qaytardığı dəyərin əksini qaytarır.Yəni verilən şərt true dəyər qaytararsa false, false dəyər qaytararsa true dəyər qayıdır.

**Assignment Operators**(Mənimsətmə operatorlar)

= 🡪 müəyyən bir dəyişənə verilən qiyməti mənimsətmək üçün istifadə edilir.

+= 🡪 müəyyən bir dəyişənə dəyişənin verilən qiymətlə cəmini mənimsətmək üçün istifadə edilir.

-= 🡪 müəyyən bir dəyişənə dəyişənin verilən qiymətlə fərqini mənimsətmək üçün istifadə edilir.

\*= 🡪 müəyyən bir dəyişənə dəyişənin verilən qiymətlə hasilini mənimsətmək üçün istifadə edilir.

/= 🡪 müəyyən bir dəyişənə dəyişənin verilən qiymətlə qismətini mənimsətmək üçün istifadə edilir.

**Misc Operators & Ternar Operators**

Ternar operator if-then-else məntiqinə əsaslanır.Əlavə kod yazmadan 1 sətr daxilində ifadə yoxlanılaraq mənimsətmə prosesi aparılır. Dəyişənimizə verilən ifadə true dəyər qaytararsa valueT, false dəyər qaytararsa valueF mənimsədilir.

Datatype variable = (expression)? valueT(if expression true) : valueF(if expression false) ;

**Control Statements | Control Flow**

Javada 3 tip control flow ifadə vardır.

* Decision Making statements və ya Selection Statements
* Loop statements
* Branching statements

**Decision Making Statements**

Decision Making Statements 2 yerə bölünür if & switch statements.

**If Statements**

If(condition) {

Statement;

}

Statement2;

Bu ifadəyə if-then ifadəsi deyilir. Əgər verilən şərt doğru olarsa if daxilindəki ifadə yerinə yetirilir və daha sonra 2-ci ifadə yerinə yetirilir. Əks təqdirdə sadəcə 2-ci ifadə yerinə yetirilir.

If(condition) {

Statement;

} else{

Statement2;

}

Bu ifadəyə if-then-else ifadəsi deyilir. Əgər verilən şərt doğru olarsa sadəcə if daxilindəki ifadə yerinə yetirilir əks təqdirdə sadəcə 2-ci ifadə yerinə yetirilir.

if(condition) {

Statement;

} else if(condition2){

Statement2;

} else if(condition3){

Statement3;

} else{

StatementN;

}

Else-if ifadəsi müəyyən bir şərt ödənmədikdə verilə biləcək digər şərtə uyğun prosesin işə salınması üçün istifadə edilir

**Switch case**

Bizim təyin edtiyimiz dəyişənin müxtəlif qiymətlərə və ya dəyərlərə bərabər olma halı yoxlanılırsa if statement yerinə switch case’dən istifadə edilir. Əgər bizim dəyişənimizə heç bir case halı uyğun olmazsa default daxilindəki əməliyyatlar işə düşür.

switch(expression) {

case value1:

Statement;

break;

case value2:

Statement2;

break;

case valueN:

StatementN;

break;

default:

StatementDefault;

break;

}

**Iteration Statements & Loop Statements**

Loop(dövr) qarşıya qoyulmuş məsələni müəyyən şərt daxilində müəyyən say qədər təkrar yerinə yetirmək üçün istifadə edilir.

**While**

while(condition) {

Statement;

}

Şərtimiz doğru olduğu müddətcə blok daxilindəki ifadə yerinə yetiriləcək.Bu da o deməkdir ki ilk olaraq hər dəfə gedib şərt yoxlanılacaq və true dəyər qayıdarsa prosess yerinə yetiriləcək.Bu səbəbdən While’a həm də ön şərtli dövr operatoru deyilir.

**Do While**

do{

Statement;

}while(condition)

Do while ilə while’ın fərqi odur ki, do while zamanı şərt yoxlanılmadan ilk başda proses 1 dəfə yerinə yetirilir daha sonra şərt yoxlanılır. Bu da o deməkdir ki verilən condition false olsa, while loop’da proses heç vaxt yerinə yetirilməyəcək amma do while’da isə proses 1 dəfə yerinə yetiriləcək. Bu səbəbdən Do While’a həm də son şərtli dövr operatoru deyilir.

**For**

for(initialization; condition; iteration){

Process;

}

For loop’u 3 hissəyə bölünür 1-ci hissə dəyişən(lər) təyin etmək üçün, 2-ci hissə şərt vermək və yoxlamaq üçün, sonuncu hissə müəyyən bir əməliyyatı yerinə yetirmək üçün istifadə olunur.Şərt ödəndiyi bütün hallarda For loop’un bloku daxilindəki proses işə düşür.

for(; condition;){

Process;

}

For üçün bu cür yazılış da mövcuddur.

**Branching Statements**

Branching Statements 3 yerə bölünür break & continue & return statements.

**Break Statements**

Müəyyən bir şərt daxilində dövrü sonlandırmaq üçün və ya switch’dən çıxmaq üçün istifadə olunan bir ifadədir.

**Continue Statements**

Müəyyən bir şərt daxilində işləndiyi yerdən sonraki kodlar yerinə yetirilmədən dövrün əvvəlinə qayıtmaq üçün istifadə edilir.

**Return Statements**

Müəyyən bir şərt daxilində methodu sonlandırmaq üçün istifadə edilir. Əgər method hər hansısa bir dəyər qaytarırsa return’ün qarşısında həmin dəyərə uyğun olan Datatype’da bir dəyər yazılmalıdır.

**For each Statements**

For each collection’lara(çoxluqlara) tətbiq olunan dövrdür. Biz for each vasitəsi ilə collectionun hər bir elementinə müraciət edə bilərik.

for(Datatype itrVariable : Collection){

Process;

}

**-Stack yaddaş və heap yaddaş**

**Wrapper type – bigInteger, bigDecimal**

**Ref out**

**Opitonals parameter**