Estructura d’un programa informàtic

1.1. Blocs d’un programa informàtic.

1.2. Tipus de dades simples i compostes.

1.3. Estructures de selecció, repetició i salt.

1.4. Tractament de cadenes.

part introducció

Quan se parla de l’estructura de un programa podem dir que primerament consisteix en un conjunt d’ordres que l’ordinador ha d’executa. L’ordinador executa les ordres (mòduls ben estructurats i separats) mentres llegeix el fitxer fins que arriba a una ordre () que introdueix l’ordinador a anar a una ubicació específica del programa i seguir fent fins arribar al final del programa.

1.1 El bloc de construcció bàsic d'un programa Java és l'afirmació de un codi. Una declaració s'acaba amb un punt i coma i pot abastar diverses línies.

Un bloc, que consisteix en 0, 1 o més declaracions, comença amb un ({) i acaba amb un (}). Els blocs són necessaris per a les definicions de classe i mètodes i es poden utilitzar en qualsevol altre lloc del programa que una declaració sigui legal.

1.2 (Els tipus de dades són diferents maneres d'interpretar aquests "zeros i uns" en funció de certes configuracions que estableixen l'espai utilitzat així com la representació aplicada per codificar i descodificar aquesta informació.

Cada tipus de dades s’identifica amb un nom i és capaç d'emmagatzemar una determinada classe d’informació, així com un rang específic de valors.)

Al moment de parla de dades simples se pot dir que són les dades indivisibles, és a dir, no es pot descompondre, per exemple un any (2021) és una dada simple ja que s’expressa amb un enter, que no es pot descompondre.

Per el contrari una dada composta es compon d’altres dades, per exemple una data, es composta perque te tres dades simples (dia:30, mes:9, any:2021)

1.3 (Una part fonamental de la creació de programes és la necessitat de canviar la seqüència d'instruccions i decidir quines instruccions executar i quines no executar. Les estructures de control de flux que permeten a l'ordinador prendre decisions són les estructures de selecció.)

Hi ha tres tipus d'estructures de selecció:

1. Estructura de selecció simple
   - L'objectiu d'aquesta estructura és decidir executar un bloc de codi o no.

Exemple:

```
if(condició){
        //codi a executar quan la condició es compleix
}else{
        //codi a executar quan la condició no es compleix
}
```

(*exemple codi*)

```
int a=15;

if(a>10){
System.out.println("Es major a 10");
}
```

2. Estructura de selecció doble
   - L'objectiu d'aquesta estructura és decidir quin bloc de codi s'ha d'executar, si el de la branca veritable o el de la branca falsa. La decisió dependrà de l'avaluació de l'expressió condicional.

Exemple:

```
if(condició1){
//codi a executar quan la condició 1 es compleix
}else if(condició2){
//codi a executar quan la condició 2 es compleix
}else if(condicióN.){
//codi a executar quan la condició N es compleix
}else{
//codi a executar quan cap de les  condicions  es compleix
}
```

(*exemple codi*)

```
 int a = 15;
 if (a > 10) {
   System.out.println("Es major a 10");
 }else if(a<10) {
   System.out.println("Es menor a 10");
 }else {
   System.out.println("Es igual a 10");
 }
```

3. Estructura de selecció múltiple

- Quan les opcions a seleccionar són massa, és difícil manejar-les amb una estructura de selecció doble. Encara que la majoria dels llenguatges de programació moderns permeten estructures de selecció niada, en alguns casos el més simple és l'ús d'una estructura de selecció múltiple.

  L'objectiu d'aquesta estructura és decidir quin bloc de codi s'ha d'executar dins *de n* possibles opcions. La decisió dependrà del valor d'una variable, que generalment hauria de ser del tipus enter.

Exemple

```
switch(variable){
case cas1 : //sentències a executar
break;
 case cas2: //sentències a executar
break;
 case casN... : //sentències a executar
break;
 default: // sentència que s'executa si no entra en cap cas
break;
}
```

(*exemple codi*)

```
 switch (opció) {
        case 1:
                resultat = num1 + num2;
                System.out.println("La suma és: " + resultat);
                break;
        case 2:
                resultadt = num1 - num2;
                System.out.println("La resta és: " + resultat);
                break;
        case 3:
                resultat = num1 * num2;
                System.out.println("La multiplicación és: " + resultat);
                break;
        case 4:
                resultat = num1 / num2;
                System.out.println("La divisió és: " + resultat);
                break;
        default:
                System.out.println("Opció no válida");
                break;
        }
```

(Les estructures de repetició permeten repetir moltes vegades un bloc de sentències. A aquestes estructures també se'ls coneix com a estructures iteratives o bucles.)

Les estructures de repetició es componen de quatre parts: la inicialització, la condició, el bloc de sentències i l'actualització.

Inicialització: permet inicializar l'estructura iterativa, normalment consisteix en la declaració i inicialització de la variable de control del bucle.
Condició: defineix la condició que s'avalua per executar el bloc de sentències.
Actualització: actualització de les variables del bucle.

Exemple:

```
inicialització;
while (condició) {
bloc-de-sentències;
actualització;
}
```

(Quan parlem de instruccions de salts se diu que provoca una modificació del flux d'execució d'un programa)

Java ofereix dues instruccions de salt:

- break
- continue

Break
Aquesta instrucció provoca la finalització d'una instrucció switch , while , do-while o for . En aquells casos en què hi hagi estructures de control repetitives, 1 break produeix la sortida immediata d'aquell bucle en el qual es trobi inclosa però no dels altres.

```
public class Exemplo1Break {
   public static void main(String[] args) {
        for (int i = 1; i <= 50; i++) {  //inicio del for
            if (i % 3 == 0 && i % 5 == 0) {
                break;     // final del for
            }
            System.out.println(i);
        } //fin del for
        System.out.println("Adeu!!!");
   }
}
```

Continue
Aquesta instrucció provoca l'execució de la següent iteració en el bucle, és a dir, se salta les instruccions que queden fins al final del bucle, i torna a l'inici de la mateixa. Si es tracta d'un bucle for torna a la zona d'increment / decrement.

```
public class Exemplo1Continue {
    public static void main(String[] args) {
        for (int i = 1; i <= 50; i++) {   //inicio del for
            if (i % 3 == 0 && i % 5 == 0) {
                continue; //següent  iteració del for
            }
            System.out.println(i);
        }  //fi del for
    }}
```

1.4 (Una cadena és una seqüència de caràcters (cadena en anglès "string"). el qual s'utilitza per a poder emmagatzemar paraules i frases. Un String és una cadena de caràcters, que no es pot modificar, llegir el seu valor, extreure els caràcters, subcadenes, etc .; i per a qualsevol modificació en Java es crearà una nova cadena.
Les cadenes són una part fonamental de la majoria dels programes, així doncs Java té diverses característiques incorporades que faciliten la manipulació de cadenes.)

Per al maneig de cadenes podem utilitzar a l'igual que en els vectors a través d'índexs, és a dir:
"informàtica"

i	n	f	o	r	m	a	t	i	c	a
0	1	2	3	4	5	6	7	8	9	10