Pau Cobacho

Arnau Adan

dimecres, 28 octubre de 2020

INTRODUCCIÓ DEL PROBLEMA

L'objectiu per aquest laboratori és implementar el codi per recrear una versió minimalista del programa "Logo", un llenguatge de codi eduacional dissenyat als anys 60. Logo consisteix en una tortuga (que figura en la pantalla) la qual pot complir determinades ordres, com per exemple girar o anar cap endavant. Les instruccions van ordenades en programes.

Al seminari 1 hem dissenyat les classes que representen les instruccions i els programes. Ara, se'ns demana implementar-les en Java. Llavors, necessitarem programar i establir les relacions pertinents entre les classes per tal de poder afegir instruccions a un programa i implementar les operacions correctes per iniciar-lo i manar operacions posteriors. El programa hauria de comportar-se tal que, al indicar el següent programa:

```
REP 4
FWD 100
ROT 90
END
```

fos capaç d'obtenir com output: FWD 100, ROT 90, FWD 100, ROT 90, FWD 100, ROT 90, FWD 100, ROT 90. O sigui, instruccions de dibuix ometent aquelles que tracten els bucles.

Per tant, necessitem definir dues classes; la primera , “Instrcution”, s’encarregarà de donar forma a les instruccions, establint la seva morfologia i brindant certes funcions per estudiar la seva validesa, el seu tipus o si representen qualsevol error conegut. La segona, “Program”, servirà per executar un seguit d’instruccions de la primera classe, sent capaç de determinar l’estat del programa (la instrucció actual), l’estat d’un bucle d’instruccions, manegar de manera correcta aquestes i verificar la precisió del programa, informant sobre els errors trobats. Per tal d’assolir les necessitats de cada classe caldrà desenvolupar els mètodes i atributs pertinents per a cadascuna:

CLASSE INSTRUCTION

La instrucció vindrà definida per un codi i un paràmete on per exemple, per “FWD 100”, “FWD” representa el codi i 100 el paràmetre d’aquest. Per tant els atributs seràn dos: **code** i **param**, el primer de variable tipus String i el segon Double (per poder introduir decimals a alguns paràmetres).

En quant als mètodes: per definir la morfologia de la instrucció implementarem un constructor que assignarà codi i paràmetre a cada instància. També implementarem getters per cada atribut, **getCode( )** i **getParam( )**, que ens serviràn més tard per obtenir informació de cada atribut de la instrucció. Pel que respecta a la validesa de cada instrucció afegirem dos mètodes; un per identificar el tipus d’error en la instrucció, **errorCode( )**, que retornarà un enter representant la categorització de l’error i un segon, **isCorrect( )** que simplement retornarà un booleà indicant l’existència d’error (true) o viceversa. Finalment, afegirem **isRepInstruction( )**, per detectar el codi “REP” i “END” en una instrucció per

més tard facilitar el treball amb bucles; retoranrà cert si aparèixen un dels dos codis en la instrucció i fals altrament, després **info( )** que donarà infromació sobre la instrucció, retorna en forma d'String el codi i paràmetre d'aquesta.

CLASSE PROGRAM

Sabent que un programa contindrà un seguit d'instruccions, necessitarem em magatzemar-les en una llista i usarem l'objecte LinkedList de Java per aquest propòsit. La classe necessitarà en tot moment estar en control del seguit d'instruccions a executar, per tant, i**nstructions** serà l'atribut que apunti a aquesta llista. També ens caldrà seguir quin és l'estat actual del programa (instrucció actual) i l'estat del loop d'instruccions actual (iteració del bucle); per aquest propòsit declararem com a atributs els enters **currentLine** i **loopIteration** que actuen com a comptadors. Finalment, voldrem anomenar d'alguna manera a al nostre programa; l'atribut del tipus string **pogramName** assoleix la tasca.

Ara, per regular el comportament d'aquesta classe necessitem establir mètodes: El constructor assignarà el nom del programa a cada instància. Pel getter, **getName( )** obtindrà el nom del programa en forma d'string. Al començar el programa, la llsta serà buida, per tant, haurem d'implementar un mètode per afegir-li instruccions, **addInstruction(c: string, p: double)**, a la qual li introduïm un codi i un paràmetre (representant una instrucció), que retorna un booleà cert al afegir-la a la llista. Un cop inicialitzem el programa o volguem tornar reiniciar-lo establirem com a l'estat atual la primera instrucció de la llista amb el mètode **restart( )**. **getNextInstruction( )** s'encarregarà de retornar la següent instrucció a seguir; si detecta "END" com a següent ordre, o ens enviarà cap a la instrucció just des-

prés del “REP” prèviament llegit usant **goToStartLoop( ),** que indica l’index d’aquesta, o bé finalitzarà el programa quan el loop indicat per “REP” ja hagi complert les iteracions es pecificades pel seu paràmetre. Aquest fet el podrem deduïr mitjançant el mètode **hasFinished( )** que retornarà un booleà indicant si el programa ha finalitzat. Finalment, per estudiar la precissió del programa esta blirem el mètode **isCorrect( )** retornant un booleà per infromar sobre la pressèn cia d’errors i **printErrors( )** per brindar pistes sobre els errors presents.

POSSIBLES SOLUCIONS ALTERNATIVES

CLASSE INSTRUCCIÓ

isRepInstruction( ): primerament haviem pensat en implementar un codi on mitjançant dos if’s s’estudiessin els cassos “END” o “REP” per retornar un true i fals altra-ment.

```
public boolean isRepInstruction(){
    boolean trigger = false;

    if(code.equals("REP")){
        trigger = true;
    }
    if(code.equals("END")){
        trigger = true;
    }
    return trigger;
}
```

Però finalment hem decidit adoptar una metodologia molt més simple i eficient:

```
public boolean isRepInstruction(){
    return code.equals("REP") || code.equals("END");

}
```

errorCode( ): Per determinar el cas d'error n.1 (el codi no figura entre els codis vàlids de logo) hem finalment optat per crear una llista on apareguin tots els codis vàlids i comprovar si el codi de la instrucció actual figura en aquesta (volent dir que és correcte):

```
public int errorCode(){
    int error_case = 0;

    String[] codes = {"REP", "FWD", "ROT", "END", "PEN"};
    boolean trigger = true;

    // Case 1: the code is not among the valid logo codes
    for(int i = 0; i < codes.length; i++){
        if (codes[i].equals(code)){
            trigger = false;
            break;
        }
    } if (trigger){
        error_case = 1;
    }
```

La millora per la funció és tracta de la keywoard "break". Aqui ens hem donat compte de que al detectar que el codi és correcte el bucle for seguia iterant, per tant, introduïnt "break" després de descartar l'error evitarà iteracions innecesaries del for, estalviant temps de compilació.

CLASSE PROGRAMA

loopIteration: hem decidit que, en comptes d'iniciar la variable amb el valor 1, indicant la primera iteració, ens convenia més inicialitzant-la amb el seu valor màxim (indicat pel paràmetre de "REP") i així acabar el programa amb **hasFinished( )** tal que:

```
public boolean hasFinished(){
    return (loopIteration == 0);
}
```

Mostrant una funció simple i elegant.

getNextInstruction( ): hem decidit posar dins d'aquesta funció el print de la inormació sobre la instrucció en comptes de posar-lo al main del projecte, tal com indica l'exemple del pdf. De la ultima manera teniem problemes per no imprimir també la informació de les instruccions "REP" i "END" ja que per aquest cas:

```
public class LogoProgram {
   public static void main ( String[] args ) {
      Program p = new Program ( "Square" ) ;
      p.addInstruction( "REP" , 4 );
      p.addInstruction( "FWD" , 100 );
      p.addInstruction( "ROT" , 90 );
      p.addInstruction( "END" , 1 );
      if( p.isCorrect() ){
         p.restart();
         while ( !p.hasFinished() ) {
            Instruction instr = p.getNextInstruction();
            System.out.println( instr.info() );
         }
      }
   }
}
```

al cridar la funció **info( )** tenint una instrucció "END" o "REP" no podrem ometre l'impressió ja que la funció mateixa exigeix un return del tipus string i no hem trobat cap manera de retornar una string "nula". Al fer, per exemple, return string null;, retornava "null". De la següent manera i excloent la ultima linea de codi del main ja som capaços d'ometre imperssió.

```
public Instruction getNextInstruction(){
    Instruction next = instructions.get(currentLine);

    if(next.isRepInstruction()){
        if(next.getCode().equals("REP")){
        double reps = next.getParam();
        loopIteration = (int)reps;
        }
        if(next.getCode().equals("END")){

            goToStartLoop();
            loopIteration -= 1;
        }
    }
    else{
        System.out.println(next.info());
    }
    currentLine += 1;
    return next;
}
```

Aqui introduïm la funció **info( )** on els codis "END" I "REP" no hi poden accedir.

printErrors( ): per aquesta funció hem decidit implementar una millora, aquesta permet informar a quina instrucció del programa apareix cada error i informar sobre el seu tipus.

```
public void printErrors(){

    LinkedList<Integer> error_n = new LinkedList<>();

    for( int i = 0; i < instructions.size(); i++){
        error_n.add(instructions.get(i).errorCode());
    }

    if(error_n.contains(1)){
        for(int i = 0; i < instructions.size(); i++){
            if(error_n.get(i) == 1){
                int index = i + 1;
                System.out.println("Instruction" + " "+ index +" "+ "ERROR: the code is
            }
        }
    }
```

Creem una llista en la qual hi figurarà el tipus enter d'error, si existeix, per cada instrucció, apareix un 0 si no existeix. Ara, la imatge mostra el cas d'error 1; on podem observar que al detectar aquest tipus d'error en la llista, busca les instruccions que el contenten i informa. Seguirà el mateix procediment per la resta de casos d'error.

CONCEPTES TEÒRICS

Esmentarem la teoria relacionada als conceptes de programació orientada a objectes que hem aplicat com a part de la solució:

Abstracció i **delegació**: percebem l'**abstracció** de la implementació que hem duut a terme al observar que, per exepmple l'objecte Program en cap moment necessita saber com funciona l'objecte Instruction, només sap que és capaç de fer el que necessita i, per tant li confia la tasca de basar el programa en instruccions. O sigui, un cop deleguem una tasca, l'altre objecte és responsable de complir-la.

Observem també el què marca la diferència entre programació i programació orientada a objectes; en aquest laboratori diferents programes (objectes) interactuen junts per solucionar un problema.

Atributs i **mètodes**: hem entés clarament com els atributs de les classes program i instruction descriuen les seves "característiques" (code i param per Instruction, indicant la seva naturalesa) i els mètodes el seu "comportament" (isCorrect( ) d'Instrcution, indicant com és comporta l'objecte en funció de les seves instanciacions).

Visibility: Hem seleccionat diferents nivells de visibilitat als membres de cada classe (privat o public) aconseguint d'aquesta manera una bona encapsulació. Els membres que no calien ser accedits des de fora de la classe els hem posat privats.

POSIBLES SOLUCIONES (conceptos teoria)

va ser mi programa cuantas clases implementado como relacionado cada meto do

CONCLUSIÓN (cómo ha ido el proyecto)(examen exhaustivo del programa vería que funciona)

va ser mi programa cuantas clases implementado como relacionado cada meto do