Funcions i procediments

Introducció

L'ús de subalgorismes o subprogrames permet dividir problemes complexos en problemes més petits que es poden tratar millor de manera separada i que permet fer proves unitàries del seu funcionament amb independència de la resta del codi.

Han de tenir una interfície clara amb la resta del codi, és a dir, tenen un únic punt d'entrada, un únic punt de sortida, reben dades d'entrada preferiblement només com a paràmetres i poden retornar un valor.

El codi extern invoca les funcions i procediments utilitzant el seu nom (identificador) i passant-li els arguments d'entrada en forma de llista de paràmetres entre parèntesis.

Quan són invocats, s'executa el codi del seu cos. Un cop finalitzat, el control es retorna a la instrucció següent a la de la seva invocació.

En funció del retorn, alguns llenguatges els classifiquen en:

- Funcions
- Procediments

D'ara endavant en direm **mètodes** als dos tipus de subprogrames.

Funcions

Retornen un valor i poden usar-se en expressions, ja que avaluen al valor retornat.

Exemple: funció que calcula i retorna el valor promig de tres valors.

```
//[tipus de retorn] [identificador] (paràmetres [tipus identificador])
float mitjana(float x, float y, float z) {
    //cos (body) de la funció
    float suma = x + y +z;
    return suma/3.0;  //instrucció de retorn amb el valor a retornar.
}
```

Per invocar aquest mètode des del programa principal:

```
// amb valors constants
float resultat1 = mitjana(2.0, 3.0, 5.0);
//amb valors provinents de variables o constants prèviament definides
float a = 2.0;
float b = 3.0;
float c = 5.0;
float resultat2 = mitjana(a, b, c);
```

Procediments

S'utilizen com les funcions però no es poden usar en expressions ja que no retornen cap valor.

```
void mostraSalutacio(String missatge) {
  System.out.println(missatge);
}
```

Pas de paràmetres

Els mètodes treballen amb els **paràmetres** com si fossin variables locals. Els valors passats a aquests paràmetres en el moment de la invocació es diuen **arguments**.

La correspondència entre paràmetres i arguments es pot establir de dues maneres:

- Correspondència posicional: s'estableix tenint en compte l'ordre d'escriptura de paràmetres i arguments.
- Correspondència per nom: en la invocació d'indica el nom del paràmetre i el seu valor (argument).

El sistema més habitual en la majoria de llenguatge és el posicional.

Hi ha també diversos mètodes d'associació entre paràmetres i arguments:

- **Pas per valor**: El valor de l'argument es copia directament al paràmetre (recordeu que els paràmetres funcionen com variables locals del mètode). El mètode no té, doncs, accés als arguments originals del subprograma invocant, sinó només a una còpia dels mateixos. Això implica que el mètode no pot modificar els valor de les variables usades com a arguments. Es diu que són **paràmetres d'entrada**, ja que només faciliten informació al mètode.
- **Pas per referència**: Els paràmetres reben com a valor, no el valor dels arguments, sinó referències a les variables originals (una referència és un apuntador que permet accedir a la ubicació de la variable en memòria). Dintre del mètode, s'utilizen en general de la mateixa manera (pot dependre del llenguatge de programació), però cal tenir present que ara els canvis en els valor dels paràmetres afecten a les variables del programa principal usades com a arguments. En aquest cas, es diu que són **paràmetres d'entrada i sortida**.

Mètodes en Java

La sintaxi de Java per definir mètodes consisteix en una primera línia amb la declaració del prototip del mètode, seguida del bloc de codi del mètode tancat entre {}.

```
/**
 * area()
 * calculates and returns area of sphere with radius passed as a parameter.
 * @param double radius the radius of the sphere.
 * @return calculated area of the sphere.
 */
    public double area(double radius) {
        return (4.0 * Math.PI *radius * radius);
    }
```

El prototip conté els següents elements:

- modificador d'accés: public, protected, private
- el tipus de dada que retorna

- l'identificador del mètode
- la llista ordenada dels paràmetres del mètode, especificant el tipus i el nom

Per invocar el mètode des de la mateixa classe, la sintaxi és la següent:

```
double resultat = area(3.0);
```

Exemples

```
import java.util.Scanner;

/**
 * Entra dades d'un rectangle i en calcula el perímetre i l'àrea.
 * @author ProvenSoft
 */
public class Geometria {

    public static void main(String[] args) {
        Scanner lector = new Scanner(System.in);
        //entrada de dades
        System.out.print("Entra la base: ");
        double baseRect = lector.nextDouble();
        System.out.print("Entra l'altura: ");
        double alturaRect = lector.nextDouble();
        //calcular el perímetre
        double perimetreRect =
                calculaPerimetreRectangle(baseRect, alturaRect);
        //calcular l'àrea
        double areaRect = calculaAreaRectangle(baseRect, alturaRect);
        //mostrar resultats
        System.out.println("El perímetre del rectangle és: "+perimetreRect);
        System.out.println("L'àrea del rectangle és: "+areaRect);
    }

    /**
     * calcula area rectangle amb la base i altura indicades
     * @param base la base del rectangle
     * @param altura l'altura del rectangle
     * @return l'àrea del rectangle
     */
    public static double calculaAreaRectangle(double base, double altura) {
        double area = base * altura;
        return area;   //return base*altura;
    }

    /**
     * calcula perímetre rectangle amb la base i l'altura indicades
     * @param base la base del rectangle
     * @param altura l'altura del rectangle
     * @return el perímetre del rectangle
     */
    public static double calculaPerimetreRectangle(double base, double altura) {
        double perimetre = 2.0 * (base + altura);
        return perimetre;
    }

}
```

Activitat

Utilitzar el depurarador (*debug*) de l'editor per executar línia a línia el programa, inspeccionant els valors de les variables i observant el flux d'execució. En particular, observar el salt des de la funció principal cap a les funcions invocades i el seu retorn. Observar també les variables que es poden inspeccionar en cada situació.

```
/**
 * Exemple de diferents tipus de funcions i procediments segons els paràmetres i
retorn
 * @author ProvenSoft
 */
public class FuncProc {

    public static void main(String[] args) {
        int a = doble(3);
        System.out.println("Valor de a:"+a);
        saluda("Peter");
        avui();
        String dia = dema();
        System.out.println(dia);
    }

    /**
     * mètode 1 paràmetre i retorna 1 valor
     */
    public static int doble(int x) {
        return 2*x;
    }

    /**
     * mètode 1 paràmetre i cap retorn
     */
    public static void saluda(String nom) {
        System.out.println("Hola "+nom);
    }

    /**
     * mètode cap paràmetre i cap retorn
     */
    public static void avui() {
        System.out.println("Avui és dimecres");
    }

    /**
     * mètode amb cap paràmetre i retorna 1 valor
     */
    public static String dema(){
        return "dijous";
    }
}
```

Àmbit, visibilitat i extensió de variables

Pas de paràmetres a mètodes

El pas de paràmetres als mètodes es fa mitjançant el **pas per valor**. Els valors que s'utilitzen en la invocació al mètode s'anomenen **arguments**, mentre que els que utilitza el mètode són els **paràmetres** (també anomenats paràmetres formals). Java fa una còpia dels arguments en els paràmetres formals.

El comportament en la invocació és diferent segons que el paràmetre tingui un tipus primitiu o referenciat. En el cas que els tipus del paràmetre sigui un tipus de dades **primitiu**, es passa una còpia del valor de l'argument. En canvi, si el tipus del paràmetre és un tipus de dades **referenciat**, el que es passa és una còpia d'una referència a l'objecte (apuntador).

Per il·lustrar aquesta qüestió, editem el programa *PassTest.java* i analitzem-ne la sortida.

```
/**
 * Classe PassTest: Comprovació que el pas de paràmetres és per valor
 */
public class PassTest {
        float ptValue;
        //Mètodes per canviar els valors actuals
        public void changeInt(int value) {
                value = 55;
        }
        public void changeStr(String value) {
                value = new String("diferent");
        }
        public void changeObjValue(PassTest ref) {
                ref.ptValue = 99.0f;
        }
        public void changeObjValue2(PassTest ref) {
                PassTest ptTemp = new PassTest();
                ptTemp.ptValue = 34;
                ref = ptTemp;
        }
        public static void main (String args []) {
                String str;
                int val;
                //Crea una instància de la classe
                PassTest pt = new PassTest();
                //Assigna un enter
                val = 11;
                //intenta canviar-lo
                pt.changeInt(val);
                //Mostra valor actual
                System.out.println("El valor de val es: "+val);
                //Assigna cadena de text
                str = new String("Hola");
                //intenta canviar-la
                pt.changeStr(str);
                //Mostra valor actual
                System.out.println("El valor de str es: "+ str);
                //Prova de canvi del valor de String
                str = new String("Adéu");
```

```
                System.out.println("El valor de str es: "+ str);
                //Assigna valor a ptValue
                pt.ptValue = 101.01f;
                //canvar el valor, a través del punter a objecte
                pt.changeObjValue(pt);
                //Mostra valor actual
                System.out.println("El valor de pt.ptValue es: "+ pt.ptValue);
                //Assigna valor a ptValue
                pt.ptValue = 101.01f;
                //s'intenta canviar el valor, reassignant el punter a objecte
                pt.changeObjValue2(pt);
                //Mostra valor actual
                //Es comprova que el valor no ha canviat, perquè no permet
canviar
                //el valor del punter a l'objecte
                System.out.println("El valor de pt.ptValue es: "+ pt.ptValue);
        }// Fi main()
}//Fi class
```

El mètode *main()* crea un tipus primitiu (*int val*), un tipus referenciat (*String str*) i una instància de la pròpia classe (*PassTest pt = new PassTest()*) per utilitzar-la com a mitjà per accedir a l'atribut *float ptValue*.

La sortida del programa és la següent:

El valor de val es: 11 El valor de str es: Hola El valor de str es: Adéu El valor de pt.ptValue es: 99.0 El valor de pt.ptValue es: 101.01

Com veieu, val no es modifica amb *pt.changeInt(val)* perquè el mètode només pot accedir i modificar la còpia local del paràmetre (*value*).

Malgrat ser una referència, *str* tampoc no es modifica amb *pt.changeStr(str)* perquè el que fa el mètode és canviar la referència (posició a on apunta) la còpia local del paràmetre value. En canvi, sí que es modifica des del main quan es canvia directament.

A través de la invocació pt.changeObjValue(pt) es modifica l'atribut ptValue perquè el paràmetre és una referència a l'objecte.

En canvi, quan fem la crida *pt.changeObjValue2(pt)*, l'atribut *ptValue* no es modifica perquè el mètode crea un nou objecte local *ptTemp* i després l'assigna al paràmetre. Amb això el que es fa és fer apuntar la còpia local del paràmetre *ref* al nou objecte local, però la referència original de la invocació no es veu afectada.

Àmbit d'una variable

L'ambit de definició d'una variable és la part del codi des d'on la variable és accessible i pot ser utilitzada en una expressió a través de l'identificador amb què ha estat declarada.

L'àmbit de les variables va des del punt en què són declarades fins al final del bloc que les conté (el bloc ve delimitat pels símbols {}).

Els blocs interns de codi tenen accés a les variables dels blocs que els contenen, llevat que es tracti de mètodes.

Totes les variables declarades dintre de mètodes són locals als mètodes. Els paràmetres dels mètodes també són locals. Per tant, no es poden utilitzar fora del mètode.

Els atributs d'un objecte declarats publics poden ser accedits des de fora del bloc de la classe, sempre usant la variable de l'objecte com a referència.

Visibilitat d'una variable

La visibilitat d'una declaració de variable és la part del codi on la variable té àmbit i l'identificador referencia la variable.

La visibilitat acostuma a coincidir amb l'àmbit, però hi ha casos en què una declaració en un bloc intern usant el mateix identificador fa invisible la declaració feta al bloc exterior. En aquest cas, preval la declaració interna, i l'externa perd visibilitat al bloc intern, tot i que es manté l'ambit.

Extensió o temps de vida d'una variable

L'extensió d'una variable fa referència a la part de codi on la variable té assignat emmagtzemament en memòria.

Les variables locals als mètodes i els seus paràmetres es creen en entrar al mètode i es destrueixen en sortir-ne. Per tant, la seva extensió és el cos del mètode.

Els atributs dels objectes tenen extensió mentre el té l'objecte i la perden quan l'objecte és destruït.

El següent exemple il·lustra els diferents àmbits de definició de variables.

```
/**
 * Exemples d'àmbits de variables
 * @author Jose
 */
public class Ambits {

    /**
     * scope: the whole class
     * visibility = scope except when being hidden by another variable with the
same name and local scope
     * extension (non-static): since object instantiation until destruction.
     * extension (static): the whole program execution.
    */
    private static String msg = "Class attribute"; //scope the whole class

    public static void main(String[] args) {
        /**
         * scope: since declaration to end of block
         * visibility = scope except when being hidden by another variable with
the same name and local scope
         * extension: since start of function execution until return
        */
        String msg = "Local in main"; //scope: main function. Hides visibility
of msg class attribute.
        System.out.println("* f1(\"Function argument\")");
        f1("Function argument");
        System.out.println("* f1(msg)");
        f1(msg);
```

```
        System.out.println("* f2()");
        f2();
        System.out.println("* f3()");
        f3();
        System.out.println("* f4(msg)");
        f4(msg);
        System.out.println("* main:println(msg)");
        System.out.println(msg);
        System.out.println("* Ambits.msg");
        System.out.println(Ambits.msg);
    }

    public static void f1(String msg) {
        //msg parameter has local scope and hides visibility of msg class
attribute in function
        System.out.println(msg);
    }

    public static void f2() {
        /**
         * scope: since declaration to end of block
         * visibility = scope except when being hidden by another variable with
the same name and local scope
         * extension: since start of function execution until return
        */
        String msg = "Local variable";
        //msg local variable has local scope and hides visibility of msg class
attribute in function
        System.out.println(msg);
    }

    public static void f3() {
        //msg class attribute keeps visibility in function
        System.out.println(msg);
    }

    public static void f4(String msg) {
        msg = "Function parameter changed";
        System.out.println(msg);
    }
}
```

Exemples

Mètode per preguntar informació a l'usuari

```
/**
 * mostra un missatge i llegeix una resposta de l'usuari
 * @param missatge el missatge a mostrar a l'usuari
 * @return la resposta de l'usuari
 */
public String entraString(String missatge) {
        Scanner lector = new Scanner(System.in);
        System.out.print(missatge);
        return lector.next();
}
```

Programa amb menú per a interactuar amb l'usuari

Exemple d'un programa de conversió entre dues monedes usant un menú per interactuar amb l'usuari i mètodes de control per respondre a les seves accions.

```
import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * Realitza conversions de moneda entre euros i dòlars. Presenta a l'usuari un
 * menú per escollir la conversió fins que demana sortir.
 *
 * @author ProvenSoft
 */
public class ConversorMoneda {

    public static void main(String[] args) {

        boolean sortir = false; //senyal per sortir

        do {
            //mostrar menu i llegir opcio
            int opcioSel = mostraMenu();
            //executar l'opció escollida
            switch (opcioSel) {
                case 0: //sortir
                    sortir = true;
                    break;
                case 1: //euros a dòlars
                    procConvEuroDolar();
                    break;
                case 2: //dòlars a euros
                    procConvDolarEuro();
                    break;
                default:
                    System.out.println("Opció no vàlida");
                    break;
            }
        } while (!sortir);

    } //final de main

    /**
     * mostra el menu del programa i llegeix l'opció escollida per l'usuari
     *
     * @return el número de l'opció escollida per l'usuari o -1 si no vàlida
     */
    public static int mostraMenu() {
        int opcio = -1;
        System.out.println("===Menú del programa de conversió euro-dòlar===");
        //mostrar opcions
        System.out.println("0. Sortir");
        System.out.println("1. Convertir d'euros a dòlars");
        System.out.println("2. Convertir de dòlars a euros");
        //llegir opció
        System.out.print("> Selecciona una opció: ");
        Scanner lector = new Scanner(System.in);
        try {
            opcio = lector.nextInt();
        } catch (InputMismatchException e ){
```

```
            opcio = -1;
        }
        return opcio;
    }

    /* ====== Mètodes de càlcul ====== */

    /**
     * converteix euros a dòlars
     *
     * @param euros els euros que s'han de convertir
     * @return la conversió a dòlars
     */
    public static double convEuroDolar(double euros) {
        final double EURO_DOLAR = 0.9;
        return euros * EURO_DOLAR;
    }

    /**
     * converteix dòllars to euros
     *
     * @param dolars els dòlars que s'han de convertir
     * @return la conversió a euros
     */
    public static double convDolarEuro(double dolars) {
        final double DOLAR_EURO = 1.0 / 0.9;
        return dolars * DOLAR_EURO;
    }

    /* ====== Mètodes de control ====== */

    /**
     * llegeix euros, converteix a dòlars i imprimeix resultat
     */
    private static void procConvEuroDolar() {
        Scanner lector = new Scanner(System.in);
        try {
            double euros = lector.nextDouble();
            double dolars = convEuroDolar(euros);
            System.out.format("%.2f euros equivalen a %.2f dòlars\n", euros,
dolars);
        } catch (InputMismatchException e) {
            System.out.println("Entrada de dades incorrecta");
        }
    }

    /**
     * llegeix dòlars, converteix a euros i imprimeix resultat
     */
    private static void procConvDolarEuro() {
        Scanner lector = new Scanner(System.in);
        try {
            double dolars = lector.nextDouble();
            double euros = convDolarEuro(dolars);
            System.out.format("%.2f dòlars equivalen a %.2f euros\n", dolars,
euros);
        } catch (InputMismatchException e) {
            System.out.println("Entrada de dades incorrecta");
        }
    }
```

```
}
```

Exercici proposat: La ruleta

Enunciat. Javadoc de la solució

Exercici proposat: Joc de pedra, paper i tissora

Implementar el joc de pedra, paper i tissora.

Cal implementar el joc d'un jugador contra la màquina.

Els jugadors juguen simultàniament, es compara el resultat de les tirades i es decideix si hi ha empat o guanya un jugador. En aquest darrer cas s'incrementa el comptador del jugador guanyador.

A cada torn s'imprimeix el resultat dels comptadors dels jugadors.

Finalitza el joc quan un jugador guanya tres tirades (configurable) o quan s'arriba a un màxim de 6 tirades (configurable).

En finalitzar el joc es mostra el marcador final amb les puntuacions de cada jugador i s'indica quin és el guanyador o si hi ha empat.

En iniciar-se el joc, l'usuari determina el nombre de tirades que cal guanyar per acabar el joc i el nombre de tirades màximes abans de finalitzar.

Després s'inicia el bucle del joc, on l'ordinador fa la tirada escollint aleatòriament i pregunta la seva tirada a l'usuari. A continuació, es determina el resultat de la tirada i s'informa del resultat i dels marcadors.

Biblioteques de funcions i classes

Les funcions en llenguatge Java sempre pertanyen a alguna classe.

Podem agrupar les funcions que anem programant i que ens seran d'utilitat en futurs programes dintre de classes.

De la mateixa manera, les classes s'organitzen en paquets (*package*) i es poden reutilitzar en futurs programes.

El disseny de les classes ha de permetre el seu ús el diversos programes sense necessitat de tocar el seu codi.

Per definir a quina classe pertany la classe que estem programant cal declarar-ho amb la paraula clau **package**.

```
package cat.proven.store.model;
public class Product {
    //...
}
```

Els paquets poden contenir subpaquets al seu interior. Es declaren separant els noms amb punts. L'estructura de paquets i subpaquets es trasllada al disc amb la mateixa estructura de directoris i subdirectoris.

Per usar la classe abans definida cal importar la seva declaració especificant el paquet on es troba.

```
import cat.proven.store.model.Product;
```

També es poden fer importacions conjuntes de totes les classes, interface, etc. d'un paquet.

```
import cat.proven.store.model.*;
```

Exemples

Classe auxiliar per a entrada i sortida estàndard i classe principal per provar el seu funcionament.

```
import java.io.PrintStream;
import java.util.Scanner;
/**
 * Auxiliar class to perform standard input and output
 * @author Jose
 */
public class IOHelper {
    public static Scanner input = new Scanner(System.in);;
    public static PrintStream output = System.out;
}

/**
 * Test input/output with IOHelper
 * @author Jose
 */
```

```
public class IOTest {
    public static void main(String[] args) {
        IOHelper.output.print("Input your name: ");
        String name = IOHelper.input.next();
        IOHelper.output.format("You wrote that your name was %s\n", name);
        IOHelper.output.print("Input your age: ");
        int age = IOHelper.input.nextInt();
        IOHelper.output.format("You wrote that your age was %d\n", age);
    }
}
```

Biblioteca de mètodes per a càlculs amb figures

Podem definir una classe amb mètodes estàtics i públics que proveeixei càlculs de perímetres i àrees de figures planes.

```
package cat.proven.geometria;

/**
 * Biblioteca de mètodes per al càlcul amb figures planes
 * @author ProvenSoft
 */
public class Geom2d {
    /**
     * calcula el perímetre d'un quadrat, donat el costat
     * @param costat el costat del quadrat
     * @return el perímetre
     */
    public static double perimetreQuadrat(double costat) {
        return 4.0 * costat;
    }

    /**
     * calcula l'àrea d'un quadrat, donat el costat
     * @param costat el costat del quadrat
     * @return l'àrea
     */
    public static double areaQuadrat(double costat) {
        return costat * costat;
    }

    /**
     * calcula el perímetre del rectangle
     * @param base la base del rectangle
     * @param altura l'altura del rectangle
     * @return el perímetre
     */
    public static double perimetreRectangle(double base, double altura) {
        return 2.0 * (base + altura);
    }

    /**
     * calcula l'àrea del rectangle
     * @param base la base del rectangle
     * @param altura l'altura del rectangle
     * @return l'àrea
     */
    public static double areaRectangle(double base, double altura) {
        return base * altura;
    }
```

```
    /**
     * calcula el perímetre de la circumferència
     * @param radi el radi de la circumferència
     * @return el perímetre
     */
    public static double perimetreCircumferencia(double radi) {
        return 2.0 * Math.PI * radi;
    }

    /**
     * calcula l'àrea del cercle
     * @param radi el radi del cercle
     * @return l'àrea
     */
    public static double areaCercle(double radi) {
        return Math.PI * radi * radi;
    }


}
```

Per provar-ne el funcionament i il·lustrar-ne l'ús, tenim la classe principal següent:

```
import cat.proven.geometria.Geom2d;

/**
 * Tester per a càlculs amb figures
 *
 * @author ProvenSoft
 */
public class GeomTester {

    public static void main(String[] args) {
        //calcular amb quadrat
        double costatQuadrat = 5.0;
        double perimetreQuadrat = Geom2d.perimetreQuadrat(costatQuadrat);
        System.out.format(
                "El perímetre del quadrat de costat %.2f és %.2f\n",
                costatQuadrat, perimetreQuadrat);
        double areaQuadrat = Geom2d.areaQuadrat(costatQuadrat);
        System.out.format(
                "L'àrea del quadrat de costat %.2f és %.2f\n",
                costatQuadrat, areaQuadrat);
        //calcular amb rectangle
        double a = 4.0, b = 3.0;
        double perimetreRectangle = Geom2d.perimetreRectangle(a, b);
        System.out.format(
                "El perímetre del rectangle de base %.2f i altura %.2f és %.2f\
n",
                a, b, perimetreRectangle);
        double areaRectangle = Geom2d.areaRectangle(a, b);
        System.out.format(
                "L'àrea del rectangle de base %.2f i altura %.2f és %.2f\n",
                a, b, areaRectangle);
        //calcular amb cercle
        double radiCercle = 3.0;
        double perimetreCercle = Geom2d.perimetreCircumferencia(radiCercle);
        System.out.format(
                "El perímetre de la circumferència de radi %.2f és %.2f\n",
                radiCercle, perimetreCercle);
```

```
        double areaCercle = Geom2d.areaCercle(radiCercle);
        System.out.format(
                "L'àrea del cercle de radi %.2f és %.2f\n",
                radiCercle, areaCercle);

    }

}
```

El resultat en pantalla seria:

```
El perímetre del quadrat de costat 5,00 és 20,00
L'àrea del quadrat de costat 5,00 és 25,00
El perímetre del rectangle de base 4,00 i altura 3,00 és 14,00
L'àrea del rectangle de base 4,00 i altura 3,00 és 12,00
El perímetre de la circumferència de radi 3,00 és 18,85
L'àrea del cercle de radi 3,00 és 28,27
```

Recursivitat

Introducció

Un mètode recursiu és aquell que es crida a sí mateix, bé directament o bé a través d'un altre mètode. En **recursió directa**, el codi del mètode f() conté una sentència que invoca a f(). En canvi, en **recursió indirecta**, el mètode f() invoca un mètode g(), el qual invoca al mètode h(), i així successivament fins que s'invoca novament el mètode f().

En la implementació d'un mètode recursiu, és necessari establir una **condició de sortida o finalització** i assegurar-se del seu compliment. Cas contrari, l'algorisme continuaria executant-se indefinidament a través de les invocacions recursives.

Iteració versus recursió

La iteració utilitza estructures repetitives i la recursió utilitza estructures de selecció. La repetició s'aconsegueix en la iteració mitjançant l'ús explícit d'estructures repetitives, mentre que en la recursió s'aconsegueix mitjançant la repetició de crides a mètodes. En ambdós casos es necessita una condició de sortida.

En general, la recursió és més lenta, atès que cal implementar repetidament el mecanisme de invocació als subprogrames o mètodes, fent una còpia dels arguments de la invocació, la qual cosa consumeix també més memòria.

Qualsevol problema que es pot resoldre recursivament, també té una solució iterativa. Utilitzarem un enfocament recursiu quan produeixi un algorisme més fàcil de comprendre i depurar i quan la solució iterativa sigui molt complexa.

Els algorismes recursius permeten resoldre problemes complexos d'una manera elegant i senzilla.

Flux de control en un mètode recursiu

Cal tenir molt present la condició de sortida. Si aquesta condició no es compleix mai, tenim recursió infinita, i les crides recursives es repeteixen fins que s'esgota la memòria disponible o el programa pateix alguna terminació anormal per algun altre error.

El flux habitual d'un mètode recursiu té el següent esquema:

- Comprovació de la condició de sortida (**cas base**)
- Invocació recursiva
- **Cas final** per finalitzar la recursió

Procediments recursius

Quan s'escriu un algorisme per realitzar una tasca, una tècnica bàsica és dividir la tasca general en subtasques més petites. Quan alguna de les subtasques és una simplificació de la mateixa tasca general, és fàcil implementar un algorisme recursiu que invoqui al mateix procediment.

Exemple: Escriure un programa que saludi amb Hip Hip Hurra, amb tants Hip com indiqui l'usuari.

```
import java.util.Scanner;
public class HipHurra {

        public static void main (String[] args) {
                int numHips;
                numHips = llegirEnter("Entra el nombre de hips: ");
                salut(numHips);
        }

        private static int llegirEnter(String missatge) {
                System.out.print(missatge);
                Scanner lector = new Scanner(System.in);
                return lector.nextInt();
        }

        private static void salut(int n) {
                if (n==0) {  //cas base
                        System.out.println("Hurra");  //cas final
                } else {  //recursió
                        System.out.print("Hip ");
                        salut(n-1);
                }
        }

}
```

Exemple: Escriure un programa que escrigui en vertical els dígits d'un nombre natural.

```
import java.util.Scanner;

public class EscriureEnterVertical {

        public static void main (String[] args) {
                System.out.print("Entra un nombre natural: ");
                Scanner lector = new Scanner(System.in);
                int num = lector.nextInt();  // nombre a escriure, a entrar per
l'usuari
                if (num>0)  {
                        printVertical(num);
                }
                else {
                        System.out.println("El nombre ha de ser positiu");
                }
        }

        static void printVertical(int n) {
                if (n<10) { // cas base
                          System.out.println(n);   // cas final
                } else {  // n te dos o mes digits
                        printVertical(n/10);
                        System.out.println(n%10);
```

```
                }
        }
}
```

Si n té una xifra (cas base) es fa el cas final (escriure n). Cas contrari, s'invoca al mateix procediment amb la part entera de la divisió de n entre 10 (el nombre original sense les unitats) i quan es retorni de les crides recursives, s'escriu el mòdul n%10 (residu de la divisió entre 10, és a dir, les unitats).

Funcions recursives

La recursivitat també es pot utilitzar amb funcions que retornen un valor.

Exemple: Escriure un programa que calculi la suma dels n primers nombres naturals.

```
import java.util.Scanner;

public class SumaNEnters {

        public static void main (String[] args) {
                System.out.print("Sumar fins a quin número: ");
                Scanner lector = new Scanner(System.in);
                int num = lector.nextInt();  // Últim enter a sumar, a entrar
per l'usuari
                if (num>0)  {
                        int suma = sumaRecursiva(num);
                        System.out.print("La suma és: "+suma);
                }
                else {
                        System.out.println("El nombre ha de ser positiu");
                }
        }

        static int sumaRecursiva(int n) {
                return ( (n==1) ? 1 : n+sumaRecursiva(n-1));
        }

}
```

La verificació de la condició de sortida es fa amb n==1. Aquest és el cas base, a partir del qual es construeix la recursió i es fan els càlculs. Si no es compleix, es fa la crida recursiva a la mateixa funció. Si es compleix la condició de sortida, s'aplica el cas final (retornar 1).

Recursivitat indirecta

Es té recursivitat indirecta quan el mètode que invoca la recursivitat no és invocat directament per sí mateix, sinó després de les invocacions a altres mètodes.

Exemple: Escriure un programa i els subprogrames per determinar si un nombre natural és parell o senar.

```
import java.util.Scanner;

public class Paritat {
```

```
        public static void main (String[] args) {
                final int MAX_NUM = 4;
                final int [] nums = {4, 3, 13, 16};
                for (int i=0; i<MAX_NUM; i++) {
                        int x = nums[i];
                        if (senar(x)) {
                                System.out.format("%d és senar\n", x);
                        } else {
                                System.out.format("%d no és senar\n", x);
                        }
                        if (parell(x)) {
                                System.out.format("%d és parell\n", x);
                        } else {
                                System.out.format("%d no és parell\n", x);
                        }
                }
        }

        static boolean senar(int n) {
                return ( (n==0) ? false : parell(n-1));
        }

        static boolean parell(int n) {
                return ( (n==0) ? true : senar(n-1));
        }

}
```

En aquest algorisme, la paritat s'obté mitjançant invocacions recursives a les funcions parell() i senar(), amb el natural anterior. La condició de sortida s'obté quan s'arriba a 0.