Esercizi sul modello Runtime di Java e sulla ricorsione

(Fondamenti di Informatica 2 – Walter Didimo)

(Soluzioni)

Esercizio 1

1) Fornisci una definizione induttiva per la funzione: pari(n): **N** → {true,false}; tale funzione restituisce il valore true se il numero n è pari e false se n è dispari.
2) Fornisci una definizione induttiva per la funzione: prodotto(n,m), che calcola il prodotto tra i numeri naturali n ed m.
3) Usando le definizioni dei punti precedenti, definire il codice della classe Numero che ha il seguente scheletro.

```
class Numero{
/* metodo ricorsivo che determina se il naturale passato come parametro è pari o dispari*/
public static boolean pari(int n){…}

/* metodo ricorsivo che calcola il prodotto dei naturali n e m */
public static int prodotto(int n, int m){…}
}
```

4) Mostrare l'evoluzione della pila di attivazione nel modello runtime di Java relativamente al programma seguente.

```
public static void main (String[] args){
      Numero.pari(3);
      Numero.prodotto (2,2);
}
```

Soluzione

1) In modo induttivo si può dire che un numero n è pari se e solo se n-1 è dispari. Il caso base è che 0 è un numero pari. In formule:

   pari(0) = true
   pari(n) = !pari(n-1) n > 0

2) In modo induttivo si può definire il prodotto(n,m) nel modo seguente:

   prodotto(n,0) = 0
   prodotto(n,1) = n
   prodotto(n,m) = n + prodotto(n,m-1)

3)

```
class Numero{
/* metodo ricorsivo che determina n è pari o dispari */
 public static boolean pari(int n){
      //pre: n>=0
      boolean ris;
      if (n == 0)
            ris=true;
      else
            ris=!(pari(n-1));
      return ris;
}
```

```
      /* metodo ricorsivo che calcola il prodotto di due naturali */
      public static int prodotto(int n, int m){
           int ris;
           if (m==0)
                 ris=0;
           else if (m == 1)
                 ris=n;
           else
                 ris=n+prodotto(n,m-1);
           return ris;
      }
    }
```

Esercizio 2 – Un oggetto della classe SequenzaDiInteri consente di rappresentare una qualunque sequenza finita di interi. La classe SequenzaDiInteri ha il seguente scheletro.

```
class SequenzaDiInteri{
  /* memorizza la sequenza di interi */
  private int[] seq;

  /* costruttore: crea un oggetto che rappresenta la sequenza a */
  public SequenzaDiInteri (int[] a){…}

  /*   calcola   la   somma   di   tutti   gli   elementi   della   sequenza,
  usando il metodo sommaRic */
  public int somma (){…}

  /*   calcola   il   prodotto   di   tutti   gli   elementi   della   sequenza,
  usando il metodo prodottoRic */
  public int prodotto (){…}

  /*   calcola   il   massimo   tra   tutti   gli   elementi   della   sequenza,
  usando il metodo massimoRic */
  public int massimo (){…}

  /* restituisce una descrizione della sequenza*/
  public String toString (){…}

  /*  metodo  ricorsivo  che  calcola  la  somma  degli  elementi  della  sequenza
  dall'indice 0 all'indice i */
  private int sommaRic (int i){…}

  /* metodo ricorsivo che calcola il prodotto degli elementi della sequenza
  dall'indice 0 all'indice i */
  private int prodottoRic (int i){…}

  /* metodo ricorsivo che calcola il massimo tra gli elementi della sequenza
  dall'indice 0 all'indice i */
  private int massimoRic (int i){…}
}
```

Scrivere il corpo dei metodi della classe SequenzaDiInteri, adottando implementazioni di metodi ricorsivi laddove specificato dal commento, e definire inoltre una classe di prova che verifica il corretto funzionamento della classe SequenzaDiInteri.

Soluzione

Viene mostrato solo il codice della classe SequenzaDiInteri.

```
class SequenzaDiInteri{

  private int[] seq;

  /* costruttore: crea un oggetto che rappresenta la sequenza a */
  public SequenzaDiInteri(int[] a){
```

```
        this.seq=new int[a.length];
        for(int i=0; i<a.length;i++)
              seq[i]=a[i];
    }

    /* calcola la somma di tutti gli elementi della sequenza, usando il metodo
    sommaRic */
    public int somma (){
          return sommaRic(this.seq.length-1);
    }

    /* calcola il prodotto di tutti gli elementi della sequenza, usando il metodo
    ricorsivo prodottoRic */
    public int prodotto (){
          return prodottoRic(this.seq.length-1);
    }

    /*  calcola  il  massimo  degli  elementi  della  sequenza,  usando  il  metodo
    ricorsivo massimoRic */
    public int massimo (){
          return massimoRic(this.seq.length-1);
    }

    /* fornisce una descrizione dell'oggetto */
    public String toString(){
         String s = "";
         for (int i=0; i<this.seq.length; i++)
           s += this.seq[i] + "  ";
         return s;
    }


    /*  metodo  ricorsivo  che  calcola  la  somma  degli  elementi  della  sequenza
    dall'indice 0 all'indice i */
    private int sommaRic (int i){
          int s;
          if(i==0)
              s = this.seq[0];
          else
              s = this.seq[i] + this.sommaRic(i-1);
        return s;
    }

    /*  metodo  ricorsivo  che  calcola  il  prodotto  degli  elementi  della  sequenza
    dall'indice 0 all'indice i */
    private int prodottoRic (int i){
          int p;
          if(i==0)
              p = this.seq[0];
          else
              p = this.seq[i] * this.prodottoRic(i-1);
          return p;
          }

    /*  metodo  ricorsivo  che  calcola  il  massimo  tra  gli  elementi  della  sequenza
    dall'indice 0 all'indice i */
    private int massimoRic (int i){
          int m;
          if(i==0)
              m = this.seq[0];
          else{
              m = this.massimoRic(i-1);
              if(this.seq[i] > m)
                    m = this.seq[i];
          }
     return m;
     }
}
```

Esercizio 3 – La classe GestioneStringhe ha il seguente scheletro:

```
class GestioneStringhe{

  /* metodo ricorsivo che calcola se una stringa è palindroma (cioè se leggerla
  da sinistra verso destra è uguale a leggerla da destra verso
  sinistra)*/
  public static boolean palindroma (String s){…}


  /* restituisce la frequenza del carattere c nella stringa s usando il metodo
  ricorsivo frequenzaRic */
  public static int frequenza (String s, char c){…}


  /* restituisce l'esito della relazione d'ordine lessicografico sulle stringhe
  s e t usando il metodo ricorsivo confrontaRic */
  public static int confronta (String s, String t){…}

  /* metodo ricorsivo che calcola la frequenza del carattere c nella stringa s,
  dal carattere in posizione p in poi */
  private static int frequenzaRic (String s, char c, int p){…}


  /* metodo ricorsivo che confronta le stringhe s e t,
  dal carattere in posizione p in poi */
  private static int confrontaRic (String s, String t, int p){…}
}
```

Scrivere il corpo dei metodi della classe GestioneStringhe e definire una classe di prova che verifica il corretto funzionamento della classe GestioneStringhe.

Soluzione

Si riporta solo il codice della classe GestioneStringhe.

```
class GestioneStringhe{

  /* metodo ricorsivo che calcola se una stringa è palindroma (cioè se leggerla
  da sinistra verso destra è uguale a leggerla da destra verso
  sinistra)*/
  public static boolean palindroma(String s) {
      boolean pal;
      if (s.length() <= 1)
            pal = true;
      else
          // una stringa s = axb è palindroma se a=b e x è palindroma
          // (a e b indicano caratteri, mentre x indica una sottostringa)
          pal=(s.charAt(0)==s.charAt(s.length()-1))
               && palindroma(s.substring(1,s.length()-1));
      return pal;
  }


  /* restituisce la frequenza del carattere c nella stringa s usando il metodo
  ricorsivo frequenzaRic */
  public static int frequenza (String s, char c){
        return frequenzaRic(s,c,0);
  }


  /* restituisce l'esito della relazione d'ordine lessicografico sulle stringhe
  s e t usando il metodo ricorsivo confrontaRic */
```

```
  public static int confronta (String s, String t){
        return confrontaRic(s,t,0);
  }


  /* metodo ricorsivo che calcola la frequenza del carattere c nella stringa s,
  dal carattere in posizione p in poi */
  private static int frequenzaRic (String s, char c, int p){
        int f;
        if(s.length() == p)
            f = 0;
        else if(s.charAt(p) == c)
            f = 1 + frequenzaRic(s,c,p+1);
        else
            f = frequenzaRic(s,c,p+1);
        return f;
  }

  /* metodo ricorsivo che confronta le stringhe s e t,
  dal carattere in posizione p in poi */
  private static int confrontaRic (String s, String t, int p){
        int k;
        if(s.length()==p && t.length()==p)
            k = 0;
        else if (s.length()==p && t.length()>p)
            k = -1;
        else if (s.length()>p && t.length()==p)
            k = 1;
        else{
            // confronta i caratteri in posizione p di s e t
            char cs = s.charAt(p);
            char ct = t.charAt(p);
            if(cs < ct)
                  k = -1;
            else if(cs >ct)
                  k = 1;
            else
                  k = confrontaRic(s,t,p+1);
         }
        return k;
  }

}
```

Esercizio 4 – Prendendo spunto dai metodi **`frequenza`** e **`confronta`**, realizzare una soluzione più efficiente per il metodo **`palindroma`** evitando di creare nuove stringhe

Soluzione

```
/* versione più efficiente del metodo palindroma */

  public static boolean palindroma(String s){
        return palindromaRic(s,0);
  }

  private static boolean palindromaRic(String s, int p){
        boolean pal;
        if (s.length() == p)
              pal = true;
        else{
              pal = (s.charAt(p)==s.charAt(s.length()-1-p) &&
              palindromaRic(s,p+1));
        }
        return pal;
  }
```