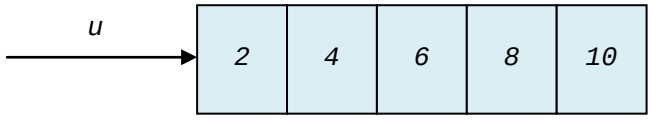
Tentamen: lösning

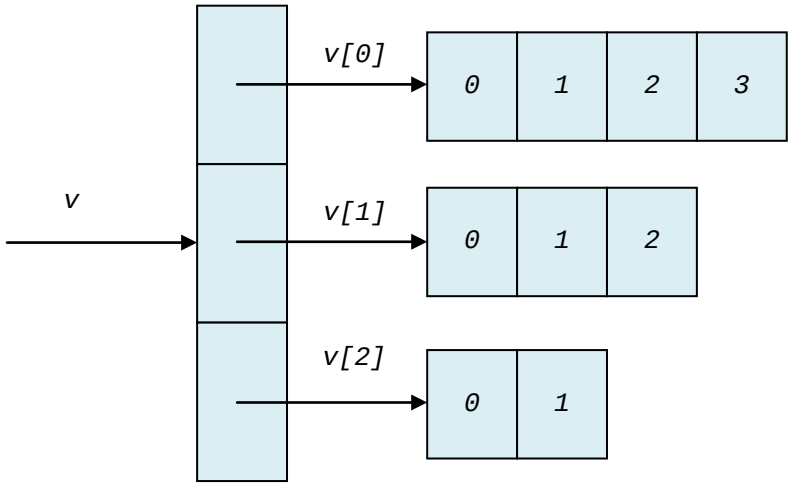
Uppgifter: lösningar

Uppgift 1 (2 poäng + 3 poäng)

a) (2 poäng)

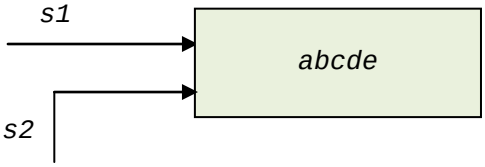


b) (3 poäng)

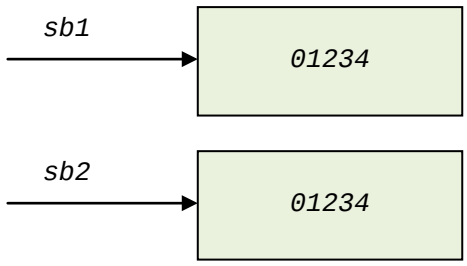


Uppgift 2 (1 poäng + 2 poäng + 2 poäng)

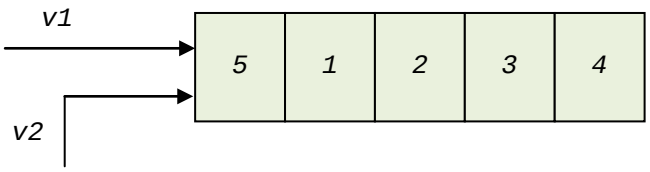
a) (1 poäng)



b) (2 poäng)



c) (2 poäng)



Uppgift 3 (3 poäng + 2 poäng + 2 poäng)

a) (3 poäng)

```
public static Point selectPoint (Point p1, Point p2)
{
    Point    p = p1;
    if (p2.distance (Point.ORIGIN) < p1.distance (Point.ORIGIN))
        p = p2;

    return p;
}
```

b) (2 poäng)

```
public static Point selectPoint (Point p1, Point p2, Point p3, Point p4)
{
    return selectPoint (selectPoint (p1, p2), selectPoint (p3, p4));
}
```

c) (2 poäng)

```
Point    p1 = new Point (2, 3);
Point    p2 = new Point (1, 4);
Point    p3 = new Point (3, 1);
Point    p4 = new Point (2, 2);

Point    p = selectPoint (p1, p2, p3, p4);
```

Uppgift 4 (3 poäng + 3 poäng + 3 poäng)

a) (3 poäng)

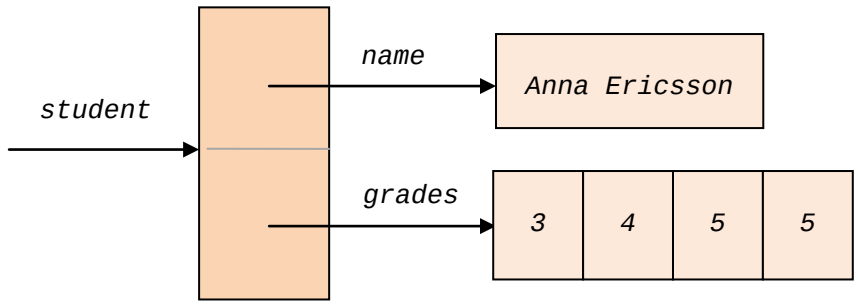
```
public String toString ()
{
```

```
    StringBuilder    s = new StringBuilder ("{" + this.name + ": [");
    int    gradeIndex = 0;
    for (gradeIndex = 0; gradeIndex < GRADE_COUNT - 1; gradeIndex++)
        s.append (grades[gradeIndex] + ", ");
    s.append (grades[gradeIndex] + "]}");

    return s.toString ();
}
```

b) (3 poäng)

```
public double averageGrade ()
{
    int    sum = 0;
    for (int gradeIndex = 0; gradeIndex < GRADE_COUNT; gradeIndex++)
        sum += grades[gradeIndex];
    double    average = (double) sum / GRADE_COUNT;

    return average;
}
```

c) (3 poäng)



Uppgift 5 (1 poäng + 2 poäng + 2 poäng + 1 poäng)

a) (1 poäng)

Klasserna `LetterSupplierA`, `LetterSupplierB` och `LetterSupplierC` implementerar gränssnittet `LetterSupplier` (klassen `LetterSupplierC` är en subklass till klassen `LetterSupplierA`, och på så sätt implementerar gränssnittet – den får metoden `lowerCaseLetter` från superklassen `LetterSupplierA`). Referenserna i vektorn `ls` är av typen `LetterSupplier`, och kan referera till objekt av alla implementationsklasser.

b) (2 poäng)

En referens av typen `LetterSupplier` aktiverar den metod som finns i det utpekade objektets definitionsklass. Detta ger följande uskrift:

```
a | A
b | B
a | C
```

c) (2 poäng)

Klassen `LetterSupplierC` är en subklass till klassen `LetterSupplierA`. Därför kan referensen `lsa1` av typen `LetterSupplierA` peka till det skapade objektet av typen `LetterSupplierC`. Superklassreferensen `lsa1` aktiverar subklassens metod `upperCaseLetter`, och följande utskrift skapas:

```
C
```

Om satsen `(1)` inkluderas uppstår ett kompileringsfel. Metoden `letter` finns inte i superklassen `LetterSupplierA`, och referensen `lsa1` kan inte aktivera denna metod. Denna metod kan aktiveras med en referens av typen `LetterSupplierC`.

d) (1 poäng)

Klassen `LetterSupplierB` är inte en subklass till klassen `LetterSupplierA`. Referensen `lsa2` av typen `LetterSupplierA` kan inte referera till det skapade objektet av typen `LetterSupplierB`. Ett kompileringsfel uppstår.

Uppgift 6 (2 poäng + 2 poäng + 5 poäng)

a) (2 poäng)

För att bestämma Fibonaccitalet på positionen n, bestäms och lagras alla föregående Fibonaccital – det finns n sådana. Minneskomplexiteten för algoritmen är:

$$m(n) = n$$

$$m(n) \in \Theta(n)$$

b) (2 poäng)

Tidskomplexiteten när det gäller additioner kan yttras med följande komplexitetsfunktion:

$$t(n) = 0, n <= 1$$

$$t(n) = n - 1, n > 1$$

$$t(n) \in \Theta(n)$$

c) (5 poäng)

```
public static long fibonacciNumber (int n)
{
    long    f = 0;
    if (n == 0)
        f = 0;
    else if (n == 1)
        f = 1;
    else
    {
        long    f1 = 0;
        long    f2 = 1;
        for (int pos = 2; pos <= n; pos++)
        {
            f = f1 + f2;
            f1 = f2;
            f2 = f;
        }
    }

    return f;
}
```

Algoritmen i den här metoden använder i värsta fall endast två extra minnesceller (`f1` och `f2`). Algoritmens minneskomplexitet är:

$$m(n) = 0, n <= 1$$

$$m(n) = 2, n > 1$$

$$m(n) \in \Theta(1)$$