Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta del 20 giugno 2016

TEMPO DISPONIBILE: 1 ora e 40 minuti

Negli esercizi proposti si utilizzano le seguenti classi:

- Classe `Object`.
  Ogni istanza di questa classe rappresenta un oggetto generico. La classe possiede un costruttore privo di argomenti. Tra i metodi della classe `Object` vi è:

  ```
  public boolean equals(Object o)
  ```

- Classe `String`.
  Ogni istanza di questa classe rappresenta una stringa di caratteri. La classe implementa l'interfaccia `Comparable` e pertanto fornisce il seguente metodo:

  `public int compareTo(String altra)`
  Confronta la stringa che esegue il metodo con quella fornita tramite l'argomento. Restituisce un valore negativo quando, in ordine lessicografico, la stringa che esegue il metodo precede quella fornita tramite l'argomento, un valore positivo quando la stringa che esegue il metodo segue quella fornita tramite l'argomento, 0 quando le due stringhe sono uguali.

  Tra gli altri metodi forniti dalla classe vi sono:

  `public int length()`
  Restituisce la lunghezza della stringa che esegue il metodo.

  `public String substring(int inizio)`
  Restituisce una stringa uguale al suffisso che, nella stringa che esegue il metodo, inizia nella posizione specificata tramite il parametro `inizio`.

  `public String substring(int inizio, int fine)`
  Restituisce una stringa uguale alla sottostringa che, nella stringa che esegue il metodo, inizia nella posizione specificata tramite il parametro `inizio` e finisce nella posizione che precede quella specificata tramite il parametro `fine`.

  Inoltre, il metodo `equals` di `Object` è ridefinito allo scopo di controllare l'uguaglianza tra stringhe.

Negli esercizi 1, 2 e 3 considerate le dichiarazioni di variabile `Object[] dati` e `String s`. Supponete di disporre di una porzione di codice alla fine della quale `dati` si riferisca a un array di riferimenti di tipo `Object` e `s` si riferisca a un oggetto di tipo `String` costruito in precedenza.
Attenzione: alcune delle posizioni dell'array potrebbero contenere il riferimento ***`null`****.*

1. Scrivete una porzione di codice per contare e visualizzare sul monitor il numero di stringhe contenute nell'array `dati` che siano uguali alla stringa riferita dalla variabile `s`.

2. Scrivete una porzione di codice per contare e visualizzare sul monitor il numero di oggetti contenuti nell'array `dati` che *siano istanze* della classe `String`.

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero di stringhe contenute nell'array `dati` che soddisfano la seguente condizione: il prefisso corrispondente alla prima metà della stringa *precede in ordine lessicografico* il suffisso rimanente (nel caso di stringhe di lunghezza dispari la metà viene calcolata per difetto). Ad esempio per la stringa `"topo"` il prefisso e suffisso da considerare sono `"to"` e `"po"`, rispettivamente, mentre per la stringa `"gatto"` sono `"ga"` e `"tto"`. Nel primo caso il prefisso segue lessicograficamente il suffisso, nel secondo lo precede.

4. Considerate le seguenti classi:

```
public class Strana {
  private static String s = "gatto";
  private int k;
  private String w;

  public Strana() {
    this("");
  }

  public Strana(String t) {
    k = t.length();
    w = t.substring(k / 2);
    s = t + "+" + s;
  }

  public int intValue() {
    return k + w.length();
  }

  public static int length() {
    return s.length();
  }
}
```

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Strana.length());  //1
    Strana s = new Strana("cavallo");
    System.out.println(s.intValue());     //2
    s = new Strana();
    System.out.println(s.intValue());     //3
    System.out.println(Strana.length());  //4
  }
}
```

Scrivete in ogni riquadro l'output prodotto dall'istruzione di stampa seguita dal commento indicato:

//1	//2	//3	//4

5. Oltre alle classi utilizzate negli esercizi precedenti, considerate una classe astratta `Alfa` che estende `Strana`, una classe concreta `Beta` che estende `Alfa` e una classe concreta `Gamma` che estende `Object`. Considerate inoltre un'interfaccia `In` implementata solo dalla classe `Beta`.

a. Nel riquadro che precede ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

- ☐ Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Alfa` non può contenere metodi concreti
- ☐ `Alfa` deve contenere almeno un metodo astratto
- ☐ `Object` è un supertipo di `In`
- ☐ Quando uno stesso metodo ha differenti segnature vi è overloading
- ☐ Quando ci sono più metodi con lo stesso nome e differenti segnature vi è overloading
- ☐ La segnatura di un metodo ne definisce la semantica
- ☐ Il contratto di un metodo ne definisce esclusivamente la sintassi
- ☐ La segnatura di un metodo ne definisce la sintassi per le chiamate
- ☐ `Strana` eredita almeno un metodo dalla propria superclasse

b. Considerate le seguenti dichiarazioni di variabile: `Object o, Strana s, Alfa a, Beta b, Gamma g, In i;`
Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente:

- ☐ `i = o`
- ☐ `o = (In) i`
- ☐ `o = new Strana()`
- ☐ `b = a`
- ☐ `i = (Alfa) s`
- ☐ `b = g`
- ☐ `b = i`
- ☐ `i = (Beta) s`
- ☐ `o = new Alfa(...)` (al posto di `...` ci sono gli argomenti richiesti)
- ☐ `o = s.toString()`

6. Considerate la dichiarazione di variabile `String[] parole` e il seguente frammento di codice:

```
int x = 9;
try {
  for (String s: parole)
    if (s != null && s.length() > 0)
      x = x - s.length();
    else
      x = 10 / x;
  x = parole[x].length();
} catch (ArithmeticException e) {
  x = 15 + x;
} catch (ArrayIndexOutOfBoundsException e) {
  x = 25 + x;
} catch (NullPointerException e) {
  x = 35 + x;
} catch (ClassCastException e) {
  x = 45 + x;
}
```

Ricordando che:

- `ArithmeticException` viene sollevata in caso di anomalie nel calcolo di operazioni aritmetiche,
- `ArrayIndexOutOfBoundsException` viene sollevata quando si tenti di accedere a una posizione inesistente in un array,
- `NullPointerException` viene sollevata quando si tenti di accedere a un oggetto tramite un riferimento `null`,
- `ClassCastException` viene sollevata quando si tenti di forzare un oggetto a una sottoclasse di cui l'oggetto non è istanza,
- `""` indica la stringa vuota,

indicate nel riquadro corrispondente, in ciascuno dei seguenti casi, il valore della variabile `x` dopo l'esecuzione:

(a) L'array riferito da `parole` contiene (nell'ordine indicato) il riferimento `null`, un riferimento a `"cane"`, un riferimento a `"gatto"`. ☐

(b) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`. ☐

(c) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`, il riferimento `null`. ☐

(d) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`, un riferimento a `"null"`. ☐

7. Considerate il seguente metodo ricorsivo. Scrivete il risultato restituito da ciascuna delle chiamate indicate nei due riquadri:

```
... int f(String x) {
  int y = x.length();
  if (y == 0)
    return 2;
  else
    return 3 * f(x.substring(0, y / 2)) - 1;
}
```

`f("oca")`	`f("papera")`

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 20 giugno 2016

TEMPO DISPONIBILE: 1 ora e 40 minuti

Negli esercizi proposti si utilizzano le seguenti classi:

- Classe `Object`.
  Ogni istanza di questa classe rappresenta un oggetto generico. La classe possiede un costruttore privo di argomenti. Tra i metodi della classe `Object` vi è:

  ```
  public boolean equals(Object o)
  ```

- Classe `String`.
  Ogni istanza di questa classe rappresenta una stringa di caratteri. La classe implementa l'interfaccia `Comparable` e pertanto fornisce il seguente metodo:

  `public int compareTo(String altra)`
  Confronta la stringa che esegue il metodo con quella fornita tramite l'argomento. Restituisce un valore negativo quando, in ordine lessicografico, la stringa che esegue il metodo precede quella fornita tramite l'argomento, un valore positivo quando la stringa che esegue il metodo segue quella fornita tramite l'argomento, 0 quando le due stringhe sono uguali.

  Tra gli altri metodi forniti dalla classe vi sono:

  `public int length()`
  Restituisce la lunghezza della stringa che esegue il metodo.

  `public String substring(int inizio)`
  Restituisce una stringa uguale al suffisso che, nella stringa che esegue il metodo, inizia nella posizione specificata tramite il parametro `inizio`.

  `public String substring(int inizio, int fine)`
  Restituisce una stringa uguale alla sottostringa che, nella stringa che esegue il metodo, inizia nella posizione specificata tramite il parametro `inizio` e finisce nella posizione che precede quella specificata tramite il parametro `fine`.

  Inoltre, il metodo `equals` di `Object` è ridefinito allo scopo di controllare l'uguaglianza tra stringhe.

Negli esercizi 1, 2 e 3 considerate le dichiarazioni di variabile `Object[] elem` e `String s`. Supponete di disporre di una porzione di codice alla fine della quale `elem` si riferisca a un array di riferimenti di tipo `Object` e `s` si riferisca a un oggetto di tipo `String` costruito in precedenza.
Attenzione: alcune delle posizioni dell'array potrebbero contenere il riferimento ***`null`****.*

1. Scrivete una porzione di codice per contare e visualizzare sul monitor il numero di stringhe contenute nell'array `elem` che siano uguali alla stringa riferita dalla variabile `s`.

2. Scrivete una porzione di codice per contare e visualizzare sul monitor il numero di oggetti contenuti nell'array `elem` che *siano istanze* della classe `String`.

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero di stringhe contenute nell'array `elem` che soddisfano la seguente condizione: il prefisso corrispondente alla prima metà della stringa *precede in ordine lessicografico* il suffisso rimanente (nel caso di stringhe di lunghezza dispari la metà viene calcolata per difetto). Ad esempio per la stringa `"topo"` il prefisso e suffisso da considerare sono `"to"` e `"po"`, rispettivamente, mentre per la stringa `"gatto"` sono `"ga"` e `"tto"`. Nel primo caso il prefisso segue lessicograficamente il suffisso, nel secondo lo precede.

4. Considerate le seguenti classi:

```
public class Strana {
  private static String s = "vipera";
  private int k;
  private String w;

  public Strana() {
    this("");
  }

  public Strana(String t) {
    k = t.length();
    w = t.substring(k / 2);
    s = t + "+" + s;
  }

  public int intValue() {
    return k + w.length();
  }

  public static int length() {
    return s.length();
  }
}
```

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Strana.length());  //1
    Strana s = new Strana("zebra");
    System.out.println(s.intValue());     //2
    s = new Strana();
    System.out.println(s.intValue());     //3
    System.out.println(Strana.length());  //4
  }
}
```

Scrivete in ogni riquadro l'output prodotto dall'istruzione di stampa seguita dal commento indicato:

//1	//2	//3	//4

5. Oltre alle classi utilizzate negli esercizi precedenti, considerate una classe astratta `Gamma` che estende `Strana`, una classe concreta `Delta` che estende `Gamma` e una classe concreta `Alfa` che estende `Object`. Considerate inoltre un'interfaccia `In` implementata solo dalla classe `Delta`.

a. Nel riquadro che precede ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

- ☐ Quando ci sono più metodi con lo stesso nome e differenti segnature vi è overloading
- ☐ La segnatura di un metodo ne definisce la semantica
- ☐ Il contratto di un metodo ne definisce esclusivamente la sintassi
- ☐ La segnatura di un metodo ne definisce la sintassi per le chiamate
- ☐ `Strana` eredita almeno un metodo dalla propria superclasse
- ☐ `Gamma` deve contenere almeno un metodo astratto
- ☐ `Object` è un supertipo di `In`
- ☐ Quando uno stesso metodo ha differenti segnature vi è overloading
- ☐ Se il codice sorgente della classe `Alfa` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Gamma` non può contenere metodi concreti

b. Considerate le seguenti dichiarazioni di variabile: `Object o, Strana s, Gamma g, Delta d, Alfa a, In i;` Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente:

- ☐ `o = (In) i`
- ☐ `d = i`
- ☐ `i = (Delta) s`
- ☐ `o = new Gamma(...)` (al posto di `...` ci sono gli argomenti richiesti)
- ☐ `o = new Strana()`
- ☐ `d = g`
- ☐ `i = (Gamma) s`
- ☐ `d = a`
- ☐ `i = o`
- ☐ `o = s.toString()`

6. Considerate la dichiarazione di variabile `String[] parole` e il seguente frammento di codice:

```
int x = 9;
try {
  for (String s: parole)
    if (s != null && s.length() > 0)
      x = x - s.length();
    else
      x = 10 / x;
  x = parole[x].length();
} catch (ArithmeticException e) {
  x = 26 + x;
} catch (ArrayIndexOutOfBoundsException e) {
  x = 36 + x;
} catch (NullPointerException e) {
  x = 46 + x;
} catch (ClassCastException e) {
  x = 56 + x;
}
```

Ricordando che:

- `ArithmeticException` viene sollevata in caso di anomalie nel calcolo di operazioni aritmetiche,
- `ArrayIndexOutOfBoundsException` viene sollevata quando si tenti di accedere a una posizione inesistente in un array,
- `NullPointerException` viene sollevata quando si tenti di accedere a un oggetto tramite un riferimento `null`,
- `ClassCastException` viene sollevata quando si tenti di forzare un oggetto a una sottoclasse di cui l'oggetto non è istanza,
- `""` indica la stringa vuota,

indicate nel riquadro corrispondente, in ciascuno dei seguenti casi, il valore della variabile `x` dopo l'esecuzione:

(a) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`, il riferimento `null`. [ ]

(b) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`, un riferimento a `"null"`. [ ]

(c) L'array riferito da `parole` contiene (nell'ordine indicato) il riferimento `null`, un riferimento a `"cane"`, un riferimento a `"gatto"`. [ ]

(d) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`. [ ]

7. Considerate il seguente metodo ricorsivo. Scrivete il risultato restituito da ciascuna delle chiamate indicate nei due riquadri:

```
... int f(String x) {
  int y = x.length();
  if (y == 0)
    return 3;
  else
    return 2 * f(x.substring(0, y / 2)) - 1;
}
```

`f("oca")`	`f("papera")`

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 20 giugno 2016

TEMPO DISPONIBILE: 1 ora e 40 minuti

Negli esercizi proposti si utilizzano le seguenti classi:

- Classe `Object`.
  Ogni istanza di questa classe rappresenta un oggetto generico. La classe possiede un costruttore privo di argomenti. Tra i metodi della classe `Object` vi è:

  ```
  public boolean equals(Object o)
  ```

- Classe `String`.
  Ogni istanza di questa classe rappresenta una stringa di caratteri. La classe implementa l'interfaccia `Comparable` e pertanto fornisce il seguente metodo:

  `public int compareTo(String altra)`
  Confronta la stringa che esegue il metodo con quella fornita tramite l'argomento. Restituisce un valore negativo quando, in ordine lessicografico, la stringa che esegue il metodo precede quella fornita tramite l'argomento, un valore positivo quando la stringa che esegue il metodo segue quella fornita tramite l'argomento, 0 quando le due stringhe sono uguali.

  Tra gli altri metodi forniti dalla classe vi sono:

  `public int length()`
  Restituisce la lunghezza della stringa che esegue il metodo.

  `public String substring(int inizio)`
  Restituisce una stringa uguale al suffisso che, nella stringa che esegue il metodo, inizia nella posizione specificata tramite il parametro `inizio`.

  `public String substring(int inizio, int fine)`
  Restituisce una stringa uguale alla sottostringa che, nella stringa che esegue il metodo, inizia nella posizione specificata tramite il parametro `inizio` e finisce nella posizione che precede quella specificata tramite il parametro `fine`.

  Inoltre, il metodo `equals` di `Object` è ridefinito allo scopo di controllare l'uguaglianza tra stringhe.

Negli esercizi 1, 2 e 3 considerate le dichiarazioni di variabile `Object[] vett` e `String s`. Supponete di disporre di una porzione di codice alla fine della quale `vett` si riferisca a un array di riferimenti di tipo `Object` e `s` si riferisca a un oggetto di tipo `String` costruito in precedenza.
Attenzione: alcune delle posizioni dell'array potrebbero contenere il riferimento ***`null`****.*

1. Scrivete una porzione di codice per contare e visualizzare sul monitor il numero di stringhe contenute nell'array `vett` che siano uguali alla stringa riferita dalla variabile `s`.

2. Scrivete una porzione di codice per contare e visualizzare sul monitor il numero di oggetti contenuti nell'array `vett` che *siano istanze* della classe `String`.

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero di stringhe contenute nell'array `vett` che soddisfano la seguente condizione: il prefisso corrispondente alla prima metà della stringa *segue in ordine lessicografico* il suffisso rimanente (nel caso di stringhe di lunghezza dispari la metà viene calcolata per difetto). Ad esempio per la stringa `"topo"` il prefisso e suffisso da considerare sono `"to"` e `"po"`, rispettivamente, mentre per la stringa `"gatto"` sono `"ga"` e `"tto"`. Nel primo caso il prefisso segue lessicograficamente il suffisso, nel secondo lo precede.

4. Considerate le seguenti classi:

```
public class Strana {
  private static String s = "ape";
  private int k;
  private String w;

  public Strana() {
    this("");
  }

  public Strana(String t) {
    k = t.length();
    w = t.substring(k / 2);
    s = t + "+" + s;
  }

  public int intValue() {
    return k + w.length();
  }

  public static int length() {
    return s.length();
  }
}
```

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Strana.length());  //1
    Strana s = new Strana("formichiere");
    System.out.println(s.intValue());     //2
    s = new Strana();
    System.out.println(s.intValue());     //3
    System.out.println(Strana.length());  //4
  }
}
```

Scrivete in ogni riquadro l'output prodotto dall'istruzione di stampa seguita dal commento indicato:

//1	//2	//3	//4

5. Oltre alle classi utilizzate negli esercizi precedenti, considerate una classe astratta `Beta` che estende `Strana`, una classe concreta `Alfa` che estende `Beta` e una classe concreta `Delta` che estende `Object`. Considerate inoltre un'interfaccia `In` implementata solo dalla classe `Alfa`.

a. Nel riquadro che precede ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

- ☐ `Beta` non può contenere metodi concreti
- ☐ `Strana` eredita almeno un metodo dalla propria superclasse
- ☐ `Beta` deve contenere almeno un metodo astratto
- ☐ Se il codice sorgente della classe `Delta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ Quando ci sono più metodi con lo stesso nome e differenti segnature vi è overloading
- ☐ La segnatura di un metodo ne definisce la semantica
- ☐ Il contratto di un metodo ne definisce esclusivamente la sintassi
- ☐ La segnatura di un metodo ne definisce la sintassi per le chiamate
- ☐ `Object` è un supertipo di `In`
- ☐ Quando uno stesso metodo ha differenti segnature vi è overloading

b. Considerate le seguenti dichiarazioni di variabile: `Object o, Strana s, Beta b, Alfa a, Delta d, In i;`
Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente:

- ☐ `o = new Beta(...)` (al posto di `...` ci sono gli argomenti richiesti)
- ☐ `o = s.toString()`
- ☐ `i = (Beta) s`
- ☐ `a = d`
- ☐ `a = i`
- ☐ `i = o`
- ☐ `o = (In) i`
- ☐ `o = new Strana()`
- ☐ `a = b`
- ☐ `i = (Alfa) s`

6. Considerate la dichiarazione di variabile `String[] parole` e il seguente frammento di codice:

```
int x = 9;
try {
  for (String s: parole)
    if (s != null && s.length() > 0)
      x = x - s.length();
    else
      x = 10 / x;
  x = parole[x].length();
} catch (ArithmeticException e) {
  x = 37 + x;
} catch (ArrayIndexOutOfBoundsException e) {
  x = 47 + x;
} catch (NullPointerException e) {
  x = 57 + x;
} catch (ClassCastException e) {
  x = 67 + x;
}
```

Ricordando che:

- `ArithmeticException` viene sollevata in caso di anomalie nel calcolo di operazioni aritmetiche,
- `ArrayIndexOutOfBoundsException` viene sollevata quando si tenti di accedere a una posizione inesistente in un array,
- `NullPointerException` viene sollevata quando si tenti di accedere a un oggetto tramite un riferimento `null`,
- `ClassCastException` viene sollevata quando si tenti di forzare un oggetto a una sottoclasse di cui l'oggetto non è istanza,
- `""` indica la stringa vuota,

indicate nel riquadro corrispondente, in ciascuno dei seguenti casi, il valore della variabile `x` dopo l'esecuzione:

(a) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`. ☐

(b) L'array riferito da `parole` contiene (nell'ordine indicato) il riferimento `null`, un riferimento a `"cane"`, un riferimento a `"gatto"`. ☐

(c) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`, un riferimento a `"null"`. ☐

(d) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`, il riferimento `null`. ☐

7. Considerate il seguente metodo ricorsivo. Scrivete il risultato restituito da ciascuna delle chiamate indicate nei due riquadri:

```
... int f(String x) {
  int y = x.length();
  if (y == 0)
    return 1;
  else
    return 3 * f(x.substring(0, y / 2)) - 1;
}
```

`f("oca")`	`f("papera")`

Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta del 20 giugno 2016

TEMPO DISPONIBILE: 1 ora e 40 minuti

Negli esercizi proposti si utilizzano le seguenti classi:

- Classe `Object`.
  Ogni istanza di questa classe rappresenta un oggetto generico. La classe possiede un costruttore privo di argomenti. Tra i metodi della classe `Object` vi è:

  ```
  public boolean equals(Object o)
  ```

- Classe `String`.
  Ogni istanza di questa classe rappresenta una stringa di caratteri. La classe implementa l'interfaccia `Comparable` e pertanto fornisce il seguente metodo:

  `public int compareTo(String altra)`
  Confronta la stringa che esegue il metodo con quella fornita tramite l'argomento. Restituisce un valore negativo quando, in ordine lessicografico, la stringa che esegue il metodo precede quella fornita tramite l'argomento, un valore positivo quando la stringa che esegue il metodo segue quella fornita tramite l'argomento, 0 quando le due stringhe sono uguali.

  Tra gli altri metodi forniti dalla classe vi sono:

  `public int length()`
  Restituisce la lunghezza della stringa che esegue il metodo.

  `public String substring(int inizio)`
  Restituisce una stringa uguale al suffisso che, nella stringa che esegue il metodo, inizia nella posizione specificata tramite il parametro `inizio`.

  `public String substring(int inizio, int fine)`
  Restituisce una stringa uguale alla sottostringa che, nella stringa che esegue il metodo, inizia nella posizione specificata tramite il parametro `inizio` e finisce nella posizione che precede quella specificata tramite il parametro `fine`.

  Inoltre, il metodo `equals` di `Object` è ridefinito allo scopo di controllare l'uguaglianza tra stringhe.

Negli esercizi 1, 2 e 3 considerate le dichiarazioni di variabile `Object[] memo` e `String s`. Supponete di disporre di una porzione di codice alla fine della quale `memo` si riferisca a un array di riferimenti di tipo `Object` e `s` si riferisca a un oggetto di tipo `String` costruito in precedenza.
Attenzione: alcune delle posizioni dell'array potrebbero contenere il riferimento ***`null`****.*

1. Scrivete una porzione di codice per contare e visualizzare sul monitor il numero di stringhe contenute nell'array `memo` che siano uguali alla stringa riferita dalla variabile `s`.

2. Scrivete una porzione di codice per contare e visualizzare sul monitor il numero di oggetti contenuti nell'array `memo` che *siano istanze* della classe `String`.

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero di stringhe contenute nell'array `memo` che soddisfano la seguente condizione: il prefisso corrispondente alla prima metà della stringa *segue in ordine lessicografico* il suffisso rimanente (nel caso di stringhe di lunghezza dispari la metà viene calcolata per difetto). Ad esempio per la stringa `"topo"` il prefisso e suffisso da considerare sono `"to"` e `"po"`, rispettivamente, mentre per la stringa `"gatto"` sono `"ga"` e `"tto"`. Nel primo caso il prefisso segue lessicograficamente il suffisso, nel secondo lo precede.

4. Considerate le seguenti classi:

```
public class Strana {
  private static String s = "topo";
  private int k;
  private String w;

  public Strana() {
    this("");
  }

  public Strana(String t) {
    k = t.length();
    w = t.substring(k / 2);
    s = t + "+" + s;
  }

  public int intValue() {
    return k + w.length();
  }

  public static int length() {
    return s.length();
  }
}
```

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Strana.length());  //1
    Strana s = new Strana("calabrone");
    System.out.println(s.intValue());     //2
    s = new Strana();
    System.out.println(s.intValue());     //3
    System.out.println(Strana.length());  //4
  }
}
```

Scrivete in ogni riquadro l'output prodotto dall'istruzione di stampa seguita dal commento indicato:

//1	//2	//3	//4

5. Oltre alle classi utilizzate negli esercizi precedenti, considerate una classe astratta `Delta` che estende `Strana`, una classe concreta `Gamma` che estende `Delta` e una classe concreta `Beta` che estende `Object`. Considerate inoltre un'interfaccia `In` implementata solo dalla classe `Gamma`.

a. Nel riquadro che precede ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

- ☐ La segnatura di un metodo ne definisce la sintassi per le chiamate
- ☐ `Strana` eredita almeno un metodo dalla propria superclasse
- ☐ Se il codice sorgente della classe `Beta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Delta` non può contenere metodi concreti
- ☐ Quando ci sono più metodi con lo stesso nome e differenti segnature vi è overloading
- ☐ La segnatura di un metodo ne definisce la semantica
- ☐ Il contratto di un metodo ne definisce esclusivamente la sintassi
- ☐ `Delta` deve contenere almeno un metodo astratto
- ☐ `Object` è un supertipo di `In`
- ☐ Quando uno stesso metodo ha differenti segnature vi è overloading

b. Considerate le seguenti dichiarazioni di variabile: `Object o, Strana s, Delta d, Gamma g, Beta b, In i;` Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente:

- ☐ `g = i`
- ☐ `i = (Delta) s`
- ☐ `g = b`
- ☐ `i = (Gamma) s`
- ☐ `o = (In) i`
- ☐ `o = new Strana()`
- ☐ `g = d`
- ☐ `o = new Delta(...)` (al posto di `...` ci sono gli argomenti richiesti)
- ☐ `o = s.toString()`
- ☐ `i = o`

6. Considerate la dichiarazione di variabile `String[] parole` e il seguente frammento di codice:

```
int x = 9;
try {
  for (String s: parole)
    if (s != null && s.length() > 0)
      x = x - s.length();
    else
      x = 10 / x;
  x = parole[x].length();
} catch (ArithmeticException e) {
  x = 48 + x;
} catch (ArrayIndexOutOfBoundsException e) {
  x = 58 + x;
} catch (NullPointerException e) {
  x = 68 + x;
} catch (ClassCastException e) {
  x = 78 + x;
}
```

Ricordando che:

- `ArithmeticException` viene sollevata in caso di anomalie nel calcolo di operazioni aritmetiche,
- `ArrayIndexOutOfBoundsException` viene sollevata quando si tenti di accedere a una posizione inesistente in un array,
- `NullPointerException` viene sollevata quando si tenti di accedere a un oggetto tramite un riferimento `null`,
- `ClassCastException` viene sollevata quando si tenti di forzare un oggetto a una sottoclasse di cui l'oggetto non è istanza,
- `""` indica la stringa vuota,

indicate nel riquadro corrispondente, in ciascuno dei seguenti casi, il valore della variabile `x` dopo l'esecuzione:

(a) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`, un riferimento a `"null"`. [ ]

(b) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`, il riferimento `null`. [ ]

(c) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"`, un riferimento a `"gatto"`. [ ]

(d) L'array riferito da `parole` contiene (nell'ordine indicato) il riferimento `null`, un riferimento a `"cane"`, un riferimento a `"gatto"`. [ ]

7. Considerate il seguente metodo ricorsivo. Scrivete il risultato restituito da ciascuna delle chiamate indicate nei due riquadri:

```
... int f(String x) {
  int y = x.length();
  if (y == 0)
    return 2;
  else
    return 2 * f(x.substring(0, y / 2)) - 1;
}
```

`f("oca")`	`f("papera")`