In blu sono indicate le risposte agli esercizi. Naturalmente per gli esercizi 1, 2, 3, in cui è richiesta la scrittura di codice, sono possibili differenti soluzioni.

In rosso sono riportati alcuni commenti relativi gli esercizi e alle soluzioni.

Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta del 12 luglio 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. Tra gli altri metodi forniti dalla classe vi sono:

  `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. Si ricordi che, in ordine lessicografico, la stringa vuota precede tutte le altre.

  `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.

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

```
int totNull = 0;
int uguali = 0;
for (Object o: dati)
  if (o == null)
    totNull++;
  else if (o.equals(s))
    uguali++;
System.out.println("null = " + totNull + " - uguali = " + uguali);
```

La chiamata o.equals(s) può essere effettuata solo dopo verificato che o non contiene il riferimento null (in tal caso non ci sarebbe l'oggetto che deve eseguire il metodo, e si avrebbe dunque una NullPointerException).
Il metodo equals può essere richiamato utilizzando direttamente il riferimento o di tipo Object. Poiché il metodo è ridefinito in String, se l'oggetto riferito da o è di tipo String sarà eseguito il metodo di String che effettua il confronto tra stringhe (chiamata polimorfa).
Non ha senso utilizzare compareTo, se si deve solo verificare l'uguaglianza.

2. Scrivete una porzione di codice per contare e visualizzare sul monitor il numero di posizioni dell'array `dati` che contengono riferimenti a istanze di classi diverse da `String`.

```
int nonString = 0;
for (Object o: dati)
  if (o != null && !(o instanceof String))
    nonString++;
System.out.println("oggetti non stringhe = " + nonString);
```

Attenzione alla richiesta del testo: contare il numero di posizioni che contengono riferimenti a istanze (=oggetti) di classi diverse da String. Pertanto le posizioni contenenti null non vanno contate in quanto non si riferiscono ad alcun oggetto. Poiché x instanceof T è sempre falsa quando x contiene null, per escludere dal conteggio le posizioni contenenti null è necessaria la condizione o != null.
Nel compito del 23 febbraio c'era un esercizio analogo. Tuttavia il testo specificava che nessuna delle posizioni dell'array conteneva null.

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero di stringhe contenute nell'array `dati` la cui lunghezza sia *inferiore* alla lunghezza della stringa riferita da `s`.

```
int piuCorte = 0;
for (Object o: dati)
  if (o instanceof String) {
    String t = (String) o;
    if (t.length() < s.length())
      piuCorte++;
  }
System.out.println("piu corte = " + piuCorte);
```

Il metodo length è disponibile in String, ma non in Object. Dunque per richiamarlo è necessario specializzare il riferimento a String, mediante la forzatura.

4. Considerate le seguenti classi:

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

  public Strana() {
    this(s);
  }

  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) {
    Strana s = new Strana();
    System.out.println(s.intValue());     //1
    System.out.println(Strana.length());  //2
    s = new Strana("cavallo");
    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
12	16	11	23

Attenzione alla differenza tra campi statici (appartengono alla classe) e non statici (appartengono ai singoli oggetti). I campi statici NON stanno negli oggetti!

5. Oltre alle classi utilizzate negli esercizi precedenti, considerate due classi concrete `Alfa` e `Beta` che estendono `Strana` e una classe astratta `Gamma` che estende `Alfa`. Considerate inoltre un'interfaccia `In` implementata dalla classe `Alfa`, ma non dalla classe `Beta`.

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

- [F] Una classe eredita tutti i metodi della superclasse, eccetto quelli dichiarati `final`
- [V] `Alfa` non può contenere metodi astratti
- [F] `Gamma` deve contenere almeno un metodo astratto
- [F] `Gamma` non possiede costruttori
- [V] I costruttori non vengono mai ereditati
- [F] Una classe può ereditare i costruttori della propria superclasse
- [F] La segnatura di un metodo ne definisce la semantica
- [F] Il contratto di un metodo ne definisce esclusivamente la sintassi
- [V] Ogni istanza di sottoclassi di `Gamma` possiede i metodi di `In`
- [V] In una classe è possibile ridefinire tutti i metodi della superclasse, eccetto quelli dichiarati `final`

I costruttori non vengono MAI ereditati! Un costruttore di una sottoclasse può chiamare esplicitamente o implicitamente un costruttore della superclasse, ma non lo eredita.

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:

- [SI] `o = g`
- [SI] `o = i`
- [SI] `s = new Strana()`
- [NO] `new Strana() = s`
- [SI] `s = new Strana((new Strana()).toString())`
- [NO] `b = s`
- [SI] `s = b`
- [SI] `i = (Gamma) s`
- [NO] `g = new Gamma(...)` (al posto di `...` ci sono gli argomenti richiesti dal costruttore)
- [SI] `b = new Beta(...)` (al posto di `...` ci sono gli argomenti richiesti dal costruttore)

Il contratto stabilisce cosa il metodo deve fare quando chiamato, quindi riguarda la semantica.

La segnatura stabilisce il "formato" con cui effettuare la chiamata (nome e argomenti), dunque riguarda la sintassi.

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

```
int x = 9;
String t = "";
try {
  for (String s: parole)
    if (s.compareTo(t) > 0)
      t = s;
  x = parole[x / t.length()].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` è vuoto, cioè non contiene alcun elemento. ArithmeticException → 24

(b) `parole` contiene `null`. NullPointerException → 44

(c) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"cane"` e un riferimento a `"gatto"`. → 5

(d) L'array riferito da `parole` contiene (nell'ordine indicato) un riferimento a `"topo"` e un riferimento a `"gatto"`. ArrayIndexOutOfBoundsException → 34

7. Considerate il seguente metodo ricorsivo e le due chiamate indicate dei riquadri. Per ciascuna di esse, nel caso la chiamata termini correttamente scrivete il risultato restituito, altrimenti scrivete ERR:

```
... int f(String x) {
  int y = x.length();
  String pref = x.substring(0, y / 2);
  String suff = x.substring(y / 2);
  if (pref == suff)
    return 1234;
  else if (pref.equals(suff))
    return f(x + x);
  else return 5678;
}
```

Attenzione alla differenza tra confronto tra riferimenti e confronto tra oggetti!

La condizione è sempre falsa, l'istruzione non viene mai eseguita

Concatenazione di stringhe: avvia una ricorsione potenzialmente infinita che si interrompe poiché a un certo punto si cerca di costruire una stringa troppo lunga (cosa succede se x si riferisce alla stringa vuota?)

f("nono")	f("nonno")
ERR	5678