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 23 febbraio 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 è:

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

  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 in cui ogni posizione contiene il riferimento a un oggetto di tipo `Object` (quindi nessuna delle posizioni contiene `null`) 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 di stringhe contenute nell'array `dati` che siano uguali alla stringa riferita dalla variabile `s`.

```
int uguali = 0;
for (Object o: dati)
  if (o.equals(s))
    uguali = uguali + 1;
System.out.println(uguali);
```

Il metodo equals, essendo già disponibile a livello di Object, può essere richiamato direttamente utilizzando il riferimento o di tipo Object. Se l'oggetto associato non è una stringa, il risultato sarà false, in quanto diverso dall'oggetto riferito da s, che è una stringa; se invece l'oggetto riferito da o è di tipo String, verrà chiamato il metodo equals di String (polimorfismo) che effettua il confronto tra stringhe. È pertanto inutile utilizzare l'operatore instanceof per controllare che l'oggetto riferito da o sia di tipo String ed effettuare una forzatura.
Per la condizione si poteva anche scrivere s.equals(o). In questo caso la chiamata non è polimorfa.
Il confronto può essere effettuato anche utilizzando il metodo compareTo della classe String, controllando che il risultato sia uguale a 0. Questa soluzione è sconsigliata in quanto prolissa. Inoltre, il metodo non è disponibile per la classe Object e richiede un parametro di tipo String. È dunque necessario controllare che o si riferisca effettivamente a una stringa ed effettuare l'opportuna forzatura.

L'inserimento di parentesi graffe inutili allunga il codice e diminuisce la leggibilità!

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

```
int cont = 0;
for (Object o: dati)
  if (!(o instanceof String))
    cont = cont + 1;
System.out.println(cont);
```

Per la condizione è preferibile l'uso dell'operatore di negazione, come sopra, alla più prolissa "o instanceof String == false". Poiché l'operatore di negazione ha precedenza più alta rispetto a instanceof, le parentesi tonde dopo ! sono necessarie.
È assolutamente necessario sapere come si negano le condizioni, soluzioni come la seguente non sono ammissibili:
if (o instanceof String) {
} else cont = cont + 1;

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor la media delle lunghezze delle stringhe contenute nell'array `dati` che, in ordine lessicografico, precedono la stringa riferita dalla variabile `s`. Nel caso l'array non contenga alcuna stringa che soddisfi tale condizione, al posto della media deve essere visualizzato un opportuno messaggio.

```
int quante = 0;
int sommaLength = 0;
for (Object o: dati)
  if (o instanceof String) {
    String stringa = (String) o;
    if (stringa.compareTo(s) < 0) {
      quante++;
      sommaLength = sommaLength + stringa.length();
    }
  }
if (quante == 0)
  System.out.println("Non e' stata trovata alcuna stringa che preceda " + s);
else
  System.out.println((double)sommaLength / quante);
```

Il metodo compareTo è disponibile nella classe String, e non nella classe Object. Pertanto non può esser richiamato mediante un riferimento di tipo Object, anche se si sia controllato, per mezzo dell'operatore instanceof, che l'oggetto associato è di tipo String. La stessa cosa vale per il metodo length. L'operazione di forzatura è dunque necessaria.
Il contratto del metodo compareTo (riportato nella prima pagina) specifica solo il segno del risultato e non il valore, che può essere un qualunque numero intero. Pertanto i controlli vanno effettuati confrontando il risultato con zero esclusivamente con uno dei tre operatori <, ==, >, scelto in base al problema in esame.
La media è un numero con la virgola, pertanto deve essere di tipo float o double. Occorre ricordare che la divisione tra interi produce sempre un risultato intero (anche se poi viene assegnato a una variabile float o double). Pertanto se, come nell'esempio, gli operandi della divisione sono di un tipo intero, occorre forzarne uno al tipo in virgola mobile, per avere una divisione in virgola mobile.
Per verificare prima del calcolo della media che non ci siano stringhe nell'array, è sbagliato controllare che sommaLength sia uguale a zero: l'array potrebbe contenere solo stringhe di lunghezza zero!

Sembra che molti studenti ignorino completamente cosa è una media e come si calcola la media di una sequenza.

Campo statico (appartiene alla classe)

Non essendoci indicato nulla, Strana estende Object

4. Considerate le seguenti classi:

```
public class Strana {
  private static String s = "gatto";
  private int k = 400;
  private int w;

  public Strana() {}

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

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

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

In mancanza di altre indicazioni, i campi numerici sono inizializzati a 0

Campi (appartengono ai singoli oggetti)

associatività a sinistra

concatenazione di stringhe

somma di int

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Strana.length());  //1
    Strana s = new Strana("cicala");
    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	606	400	8

Attenzione alla differenza tra campi e campi statici!

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:

F Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore

F `Beta` deve fornire l'implementazione di tutti i metodi di `Alfa`

Beta deve fornire l'implementazione dei metodi astratti di Alfa

F `Gamma` può contenere un metodo astratto

V `Beta` deve fornire l'implementazione dei metodi di `In`

V La classe `Alfa` possiede un costruttore

Ogni classe possiede un costruttore (se non è specificato viene aggiunto dal compilatore)

F Se il codice sorgente della classe `Alfa` non contiene un costruttore allora il compilatore segnala un errore

V Ogni istanza di `Strana` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)

F Ogni istanza di `Strana` contiene esattamente 3 campi (oltre a quelli ereditati dalla superclasse)

V I costruttori di `Strana` richiamano il costruttore senza argomenti della superclasse

F Non esistono superclassi di `Strana`

Object

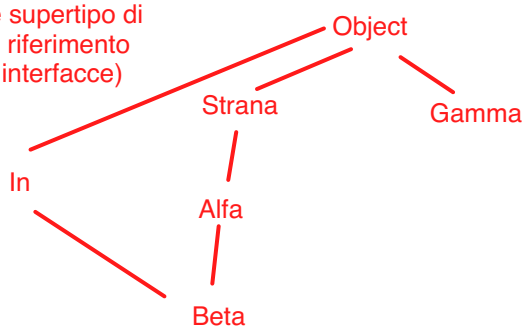
Poiché i costruttori non iniziano né con this(...) né con super(...) il compilatore aggiunge automaticamente all'inizio di essi una chiamata al costruttore privo di argomenti della superclasse

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:

Object è supertipo di tutti i tipi riferimento (incluse interfacce)



SI `o = i`

SI `i = (In) o`

SI `o = new Object()`

SI `a = b`

NO `s = (Alfa) i`

NO `g = b`

SI `i = b`

SI `s = (Beta) i`

NO `s = g`

NO `g = s`

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

```
int x = 4;
try {
  for (Object o: oggetti)
    x = o instanceof String ? ((String)o).length() : x / 2;
  String s = (String) oggetti[x / x];
  x = s.length();
} catch (ArithmeticException e) {
  x = 31 + x;
} catch (ArrayIndexOutOfBoundsException e) {
  x = 81 + x;
} catch (NullPointerException e) {
  x = 91 + x;
} catch (ClassCastException e) {
  x = 51 + 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 `oggetti` contiene (nell'ordine indicato) il riferimento a un oggetto costruito mediante `new Object()`, un riferimento a `""`, `null`. ArithmeticException → 31

(b) `oggetti` contiene `null`. NullPointerException → 95

(c) l'array riferito da `oggetti` contiene (nell'ordine indicato) il riferimento a un oggetto costruito mediante `new Object()`, e i riferimenti a `""` e a `"null"`. → 0

(d) l'array riferito da `oggetti` contiene (nell'ordine indicato) il riferimento a `"gatto"`, il riferimento a un oggetto costruito mediante `new Object()`, `null`. ClassCastException → 52

È una stringa!

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

```
... int f(int x) {
  if (x * x == x + x)
    return x + 2;
  else
    return 3 * f(x / 2);
}
```

f(4)	f(10)
12	36