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 1° febbraio 2016

TEMPO DISPONIBILE: 1 ora e 40 minuti

Negli esercizi proposti si utilizzano le seguenti classi:

- Classe astratta `Figura`.
  Ogni oggetto della classe rappresenta una figura geometrica nel piano. La classe possiede un costruttore privo di argomenti.
  Tra i metodi della classe `Figura` vi sono:

  `public abstract double getArea()`
  restituisce l'area della figura che esegue il metodo;

  `public boolean haPerimetroMinore(Figura altra)`
  restituisce `true` se e solo se il perimetro della figura che esegue il metodo è minore di quello della figura di cui viene fornito il riferimento tramite l'argomento.

- Classi `Rettangolo`, `Cerchio`, `Quadrato`.
  Gli oggetti rappresentano, rispettivamente, rettangoli, cerchi e quadrati. `Rettangolo` e `Cerchio` estendono direttamente `Figura`, mentre `Quadrato` estende direttamente `Rettangolo`.

  Tra i metodi forniti dalla classe `Rettangolo` vi sono:

  `public double getBase()`
  `public double getAltezza()`
  restituiscono, rispettivamente, la base e l'altezza del rettangolo che esegue il metodo.

  L'unico costruttore fornito da `Rettangolo` ha due argomenti:

  `public Rettangolo(double b, double a)`
  costruisce un oggetto che rappresenta un rettangolo, la cui base e la cui altezza hanno le lunghezze fornite, rispettivamente, tramite il primo e il secondo parametro.

Attenzione: nello svolgere gli esercizi fate riferimento a quanto indicato sopra e non all'implementazione delle classi, che è privata e dunque non accessibile al di fuori del codice di ciascuna delle classi stesse.
Per le classi elencate in precedenza, potete servirvi esclusivamente dei metodi indicati sopra.

Negli esercizi 1, 2 e 3 considerate le dichiarazioni di variabile `Figura[] ar` e `Rettangolo r`. Supponete di disporre di una porzione di codice alla fine della quale `ar` si riferisca a un array in cui ogni posizione contiene il riferimento a un oggetto di tipo `Figura` (quindi nessuna delle posizioni contiene `null`) e `r` si riferisca a un oggetto di tipo `Rettangolo` costruito in precedenza.

1. Scrivete una porzione di codice per calcolare e visualizzare sul monitor la somma delle *aree* delle figure contenute nell'array `ar` il cui *perimetro* sia minore di quello del rettangolo riferito da `r`.

```
double sommaAree = 0;
for (Figura f: ar)
  if (f.haPerimetroMinore(r))
    sommaAree = sommaAree + f.getArea();
System.out.println(sommaAree);
```

Il metogo getArea è disponibile nella classe Figura e dunque può essere richiamato utilizzando il riferimento f di tipo Figura.
È preferibile l'uso del ciclo for-each, più compatto rispetto all'uso di un ciclo for con indice (da utilizzare invece quando sia necessario menzionare esplicitamente le posizioni in un array).
La variabile sommaAree deve essere dichiarata di tipo double (non int, visto che getArea restituisce double) e inizializzata a 0 (visto che il valore iniziale rappresenta la somma di 0 elementi che, appunto, vale 0).
Come specificato nel testo, la variabile r è già dichiarata e si riferisce a un oggetto di tipo Rettangolo: è inutile copiarne il contenuto in un'altra variabile da utilizzare nei confronti. È inoltre inutile forzare r al tipo Figura per fornire l'argomento al metodo haPerimetroMinore: poiché Figura è supertipo di Rettangolo vi è già una promozione automatica.

L'inserimento di parentesi graffe inutili allunga il codice e diminuisce la leggibilità!
La scrittura "f.haPerimetroMinore(r) == true" nella condizione è inutilmente ridondante e rappresenta un pessimo stile: "f.haPerimetroMinore(r)" è già di tipo boolean ed è proprio la condizione da controllare.

2. Scrivete una porzione di codice per calcolare e visualizzare sul monitor la somma delle *aree* dei rettangoli (inclusi i quadrati) contenuti nell'array `ar` il cui *perimetro* sia minore di quello del rettangolo riferito da `r`.

```
double sommaAreeRettangoli = 0;
for (Figura f: ar)
  if (f instanceof Rettangolo && f.haPerimetroMinore(r))
    sommaAreeRettangoli = sommaAreeRettangoli + f.getArea();
System.out.println(sommaAreeRettangoli);
```

Oltre a quanto osservato per l'esercizio precedente, poiché Quadrato è una sottoclasse di Rettangolo, ogni istanza di Quadrato è anche un'istanza di Rettangolo. È dunque inutile aggiungere il controllo "f instanceof Quadrato".
Visto che l'area va aggiunta alla somma se la figura è un Rettangolo E il suo perimetro è minore di quello della figura riferita da r, è bene scrivere la congiunzione (and) delle due condizioni, come nella soluzione proposta, piuttosto che utilizzare due selezioni innestate.

Attenzione a come si scrive instanceof!

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero di rettangoli (inclusi i quadrati) contenuti nell'array `ar` che hanno base minore della base del rettangolo riferito da `r`.

```
int contaRettangoli = 0;
for (Figura f: ar)
  if (f instanceof Rettangolo) {
    Rettangolo rett = (Rettangolo) f;
    if (rett.getBase() < r.getBase())
      contaRettangoli++;
  }
System.out.println(contaRettangoli);
```

Visto che il valore della base del rettangolo riferito da r potrebbe essere utilizzato più volte, la soluzione può essere migliorata calcolando tale valore prima dell'esecuzione del ciclo e memorizzandolo in una variabile, da utilizzare poi per il confronto.
Il metodo getBase è disponibile per la classe Rettangolo, ma non per la classe Figura. Pertanto non può essere invocato tramite un riferimento di tipo Figura (anche se si sia verificato che l'oggetto associato è di tipo Rettangolo). È dunque necessaria un'operazione di forzatura per specializzare il riferimento (naturalmente dopo avere controllato che l'oggetto sia effettivamente un Rettangolo).

4. Considerate le seguenti classi:

```
public class Strana extends Rettangolo {
  private String s;
  private static int k = 2;

  public Strana(String t) {
    super(t.length(), t.length());
    s = t;
    k = k + t.length();
  }

  public double getArea() {
    return super.getArea() - s.length();
  }

  public static int getStatico() {
    return k;
  }
}
```

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Strana.getStatico());  //1
    Rettangolo r = new Rettangolo(3, 5);
    System.out.println(r.getArea());          //2
    r = new Strana("gatto");
    System.out.println(r.getArea());          //3
    System.out.println(Strana.getStatico());  //4
  }
}
```

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

//1	//2	//3	//4
2	15	20	7

5. Oltre alle classi utilizzate negli esercizi precedenti, considerate una classe concreta `Gamma` che estende `Figura` e una classe concreta `Delta` che estende `Strana` e implementa un'interfaccia `In`.

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

- V — Ogni istanza di `Strana` contiene esattamente un campo (oltre a quelli ereditati dalla superclasse)
- F — Ogni istanza di `Strana` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- F — `Gamma` deve fornire l'implementazione del metodo `haPerimetroMinore`
- V — `Gamma` deve fornire l'implementazione del metodo `getArea`
- F — `Gamma` deve fornire l'implementazione del metodo `getBase`
- F — Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
- V — Se il codice sorgente della classe `Delta` non contiene un costruttore allora il compilatore segnala un errore
- F — Nel codice di `Delta` è possibile accedere al campo `s` dichiarato in `Strana`
- F — `Delta` deve fornire l'implementazione del metodo `getBase`
- V — `Delta` deve fornire l'implementazione dei metodi di `In`

b. Considerate le seguenti dichiarazioni di variabile:

```
Figura f, Rettangolo r, Gamma g, Delta d, Strana s, In i;
```

Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente:

- SI — `f = new Rettangolo(s.getAltezza(), s.getBase())`
- SI — `r = new Rettangolo(r.getAltezza(), r.getBase())`
- SI — `r = (Rettangolo) f`
- NO — `f = new Figura()`
- NO — `g = (Gamma) i`
- SI — `d = (Delta) i`
- NO — `d = s`
- SI — `s = d`
- SI — `r = (Delta) i`
- NO — `f = (Gamma) i`

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

```
int x = 4;
try {
  for (Figura f: figure)
    x = f instanceof Rettangolo ? (int)((Rettangolo) f).getBase() : x / 2;
  Quadrato r = (Quadrato) figure[x / x];
  x = (int) r.getArea();
} catch (ArithmeticException e) {
  x = 22 + x;
} catch (ArrayIndexOutOfBoundsException e) {
  x = 42 + x;
} catch (NullPointerException e) {
  x = 52 + x;
} catch (ClassCastException e) {
  x = 32 + 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,

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

(a) `figure` contiene `null`. NullPointerException — 56

(b) l'array riferito da `figure` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano un quadrato di lato 2, un quadrato di lato 1, un cerchio di raggio 3. ArithmeticException — 22

(c) l'array riferito da `figure` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano un quadrato di lato 2, un cerchio di raggio 3, un quadrato di lato 1. ClassCastException — 33

(d) l'array riferito da `figure` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano un cerchio di raggio 1, un quadrato di lato 2. — 4

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

```
... int f(int x) {
  if (x <= 1)
    return 2;
  else if (x % 2 == 0)
    return 2 * f(x / 2);
  else
    return f(x + 1) + 1;
}
```

f(4)	f(6)
8	18