

Cognome**Nome****Matricola****1 (12 punti)**Data la classe **Automobile**:

```

public class Automobile {
    public static int
    autoCircolanti;
    private double serbatoio;
    private String colore;
    private String tipo;
    /* le costanti indicano il consumo in Km al
    litro di carburante
    private static final int ALTO=5;
    private static final int
    MEDIO=10;
    private static final int
    BASSO=15;

    public Automobile() {
        serbatoio = 0.0;
        colore = "";
        tipo = "";
        autoCircolanti++;
    }

    public Automobile(double dato) {
        serbatoio = dato;
        colore = "";
        tipo = "";
        autoCircolanti++;
    }

    public void setColore(String c)
    {
        colore = c;
    }

    public void setTipo( String t) {
        tipo = t;
    }

    public String getColore() {
        return colore;
    }

```

```

    public String getTipo() {
        return tipo;
    }

    public void rifornire(double quanto) {
        this.serbatoio += quanto;
    }

    public double livello() {
        return serbatoio;
    }

    public double consumare(double
    quanto){
        return serbatoio -= quanto;
    }

    public double consumare(char t, int
    k){
        double quanto;
        switch ( t ) {
            case 'a': quanto = k/ALTO; break;
            case 'm': quanto = k/MEDIO; break;
            case 'b': quanto = k/BASSO; break;
            default: quanto = k/MEDIO;
        }/* fine switch
        return serbatoio -= quanto;
    }/* fine consumare

    public String toString() {
        return "Auto " + tipo + "; colore " +
        colore + "; serbatoio " +
        this.livello();
    }
}/* fine classe Automobile

```

Le richieste 1, 10 e 15 sono relative al completamento o modifica del codice della classe **Automobile** tutte le altre sono da intendersi come codice di una diversa classe. Ad esempio come istruzioni del metodo **main** di una classe **TestAutomobile**.

1. Completare il codice dei metodi **setTipo()** **getColore()** e **getTipo()**
2. Istanziare un oggetto della classe **Automobile** di nome **autoRossa** disponendo un serbatoio di 60 litri:
Automobile autoRossa = new Automobile(60);
3. Inizializzare il campo **colore** della variabile **autoRossa** al valore "rosso":
autoRossa.setColore("rosso");
4. Inizializzare il campo **tipo** della variabile **autoRossa** al valore "Fiat Marea":
autoRossa.setTipo("Fiat Marea");
5. Istanziare un oggetto della classe **Automobile** di nome **autoBlu** con il costruttore di default:
Automobile autoBlu = new Automobile();
6. Inizializzare il campo **serbatoio** della variabile **autoBlu** rifornendo 30 litri di carburante:
autoBlu.rifornire(30.0);
7. Inviare alla variabile **autoRossa** il messaggio **consumare** 40.5 litri di carburante:
autoRossa.consumare(40.5);
8. Controllare il livello del carburante nel serbatoio di entrambe le variabili **autoRossa** e **autoBlu** invocando il metodo opportuno e calcolandone il valore di ritorno allo stato attuale

autoRossa.livello()	19.5
autoBlu.livello()	30.0
9. Ricalcolare i livelli del carburante di entrambe le variabili **autoRossa** e **autoBlu** dopo aver eseguito:
autoBlu.rifornire(autoRossa.livello());
autoRossa.rifornire(autoBlu.livello());

autoRossa.livello()	69.0
autoBlu.livello()	49.5
10. Sovraccaricare il metodo **consumare()** della classe **Automobile** con un metodo che accetti in input 2 valori, di tipo char e di tipo int, che indicano rispettivamente il livello di consumo indicativo e il numero di Km percorsi. Il livello di consumo viene indicato convenzionalmente con il carattere 'a' per indicare un consumo alto, 'm' uno medio e 'b' uno basso. Si realizzi il codice usando un'istruzione switch che seleziona in funzione del carattere di input l'istruzione da eseguire per ottenere il consumo effettivo in litri di carburante facendo uso delle costanti **ALTO**, **MEDIO** e **BASSO** definite nella classe. Il caso di default può essere definito come il caso di consumo medio. Si segua lo schema dato nel codice della classe.
11. Scrivere le corrette dichiarazioni per l'apertura dei canali di input e output usando la libreria **prog.io**
ConsoleInputManager in = new ConsoleInputManager();
ConsoleOutputManager out = new ConsoleOutputManager();

12. Scrivere le istruzioni che inviano un opportuno prompt e acquisiscono i valori del livello di consumo e i km percorsi, con cui inizializzare variabili **catConsumo** e **km** da dichiararsi in modo opportuno

```
char catConsumo = in.readChar("categoria consumo auto rossa: ");
int km = in.readInt("km da percorrere: ");
```

13. Invocare il metodo **consumare(catConsumo, km)** appena definito, per entrambe le variabili **autoRossa** e **autoBlu** usando i valori appena raccolti per la variabile **autoRossa** e altri da richiedere per la variabile **autoBlu**

```
autoRossa.consumare(catConsumo, km)
catConsumo = in.readChar("categoria consumo auto blu: ");
km = in.readInt("km da percorrere: ");
autoBlu.consumare(catConsumo, km)
```

14. Calcolare il livello di carburante per le variabili **autoRossa** e **autoBlu** nell'ipotesi che l'utente alle istruzioni precedenti abbia digitato 'a' e 550 per la variabile **autoRossa**, 'b' e 100 per la variabile **autoBlu**

```
autoRossa.livello()      ⇒    -41.0
autoBlu.livello()        ⇒    43.5
```

15. Sovraccaricare il costruttore della classe **Automobile** con un costruttore che riceva in input tre valori appropriati per inizializzare i tre campi **serbatoio colore tipo**

```
public Automobile(
    double serbatoioIniziale, String tipo, String colore){
    this.serbatoio = serbatoioIniziale;
    this.tipo = tipo;
    this.colore = colore;
    autoCircolanti++;
}
```

16. Istanziare due nuove variabili di classe dal nome **autoMetallizzata** con i valori 43.7, "Toyota Corolla", "grigio met" e **miaAuto** con 70.4, "Fiat Tipo", ""

```
Automobile autoMetallizzata =
    new Automobile(43.7, "Toyota Corolla", "grigio met");
Automobile miaAuto = new Automobile(70.4, "Fiat Tipo", "");
```

17. Interrogare la variabile **autoMetallizzata** per ottenerne il tipo

```
autoMetallizzata.getTipo()    ⇒ Toyota Corolla
```

18. Scrivere lo stato attuale dopo aver eseguito (ricordarsi di completare l'ultima istruzione)

```
out.println(autoMetallizzata);
out.println(miaAuto);
out.println("numero macchine circolanti: " +
    Automobile.autoCircolanti)
```

```
Auto Toyota Corolla; colore grigio met; serbatoio 43.7
Auto Fiat Tipo; colore grigio met; serbatoio 70.4
numero macchine circolanti: 4
```

2 (4 punti)

Scrivere un'istruzione condizionale che controlli che la richiesta

consumare(catConsumo, km) inviata all'oggetto **autoRossa** possa essere

soddisfatta e in caso contrario invii a stampa su video un segnale "macchina ferma: mancanza di carburante"

```
if (autoRossa.consumare(catConsumo, km)<0)
    out.println(autoRossa.getTipo() +
        "ferma: mancanza di carburante");
```

Riscrivere il metodo `livello()` utilizzando l'operatore condizionale ternario (`? :`) che assegni il valore `0` al serbatoio se questo risulta essere negativo (perché ad esempio una precedente richiesta di consumo era troppo elevata per l'effettivo contenuto del serbatoio)

```
public double livello() {
    return serbatoio = serbatoio < 0? 0 : serbatoio;
}
```

3 (.5 punto)

Dire qual è la caratteristica di Java che consente di definire due costruttori `Automobile`:

sovraccaricamento o overloading

Dire come si distinguono i due costruttori: **dai parametri (numero, tipo e ordine)**

4 (2 punti)

Assumendo la dichiarazione:

```
Random rand = new Random();
```

Indicare il range dei valori delle seguenti dichiarazioni:

```
rand.nextInt() % 10; [-9, 9]
```

```
(int) (Math.random() * 5); [0, 4]
```

Inoltre scrivere un'istruzione per produrre valori pseudo-casuali nell'intervallo:

`[-1, 5]` usando l'oggetto `rand` **`rand.nextInt() % 4 + 2`**

oppure **`Math.abs(rand.nextInt() % 7) - 1`**

oppure **`rand.nextInt(7) - 1`**

`[6, 12]` usando il metodo `random()` della classe `Math`

`(int) Math.random() * 7 + 6`

5 (1.5 punti)

Esprimere in linguaggio Java la seguente condizione, usando gli operatori di relazione e quelli logici:

il numero `n` deve essere **maggiore di 5 ma non di 10**

`n > 5 && !(n > 10)` o anche: `5 < n && n <= 10`

Esprimere in linguaggio Java la negazione della condizione precedente senza introdurre l'operatore di negazione (applicare la legge di De Morgan).

`!(5 < n && n <= 10)` equivale a: `n <= 5 || n > 10`

6 (2 punti)

Data la stringa:

```
String riga=new String("I cipressi che a Bolgheri alti e schietti");
```

calcolare l'output delle istruzioni:

```

riga.length();           41
riga.substring(4, 10).length(); 6
riga.substring(4, 10).toUpperCase();  PRESSI
riga.replace('e', 'a').substring(4, 10); prassi
String nuova = riga.substring(15, 27).replace('a', 'A');
System.out.println (nuova);
                        A Bolgheri A
System.out.println (riga);
                        I cipressi che a Bolgheri alti e schietti

```

7 (1 punti)

Indicare l'ordine di valutazione degli operatori nelle seguenti espressioni che assumiamo corrette, mettendo il numero corrispondente sotto al simbolo dell'operatore (considerate anche l'operatore dot).

```

x = a = b-- * ++a;
  5   4   1   3   2

x = numero.isDigit() && !finito
  5           2           1   4   3

```

8 (2 punti)

Date le variabili: `int a = -2, b = 5;` eseguire i due blocchi di istruzioni separatamente:

<code>a = a + b;</code> <code>b += a + --b;</code>	<code>a = b-- * ++a;</code> <code>b += a - b--;</code>
---	---

E calcolare il valore di a, b:

<code>a: 3</code> <code>b: 12</code>	<code>a: -5</code> <code>b: -5</code>
---	--

9 (4 punti)

Data l'espressione booleana `numero > 5 || !finito` compilarne la tabella di verità.

numero > 5	finito	!finito	numero > 5 !finito
F	F	V	V
F	V	F	F
V	F	V	V
V	V	F	V

Specificare una possibile coppia di valori delle variabili `numero` e `finito` per rendere falsa la condizione:

numero = 4 e finito = vero

Se l'espressione fosse usata come condizione in un ciclo `while` (si entra e si resta nel ciclo se la condizione è vera) dire quante volte viene eseguito il blocco d'istruzioni: **3 volte**

```
int numero = 7;
```

```
boolean finito = false;  
while (<condizione>) {  
    <istruzioni>; --numero; finito = !finito}
```

Infatti la condizione di ingresso sarebbe valutata:

numero=7	quindi	numero>5	vero	 	finito=falso	<condizione>=V
numero=6	quindi	numero>5	vero	 	finito=vero	<condizione>=V
numero=5	quindi	numero>5	falso	 	finito=falso	<condizione>=V
numero=4	quindi	numero>5	falso	 	finito=vero	<condizione>=F

e nel ciclo non si rientra.