Corso di laurea in Informatica AA 2005/06 Programmazione, M.A. Alberti

Compitino II - 14 dicembre 2005

Cognome **Nome**

Matricola

1

Data la classe `Robot`.

```
public class Robot {
   int final POTMIN = 3;
   static int numRobot;
   int potenza;
   int direzione;
   Point posizione;

   Robot() {
      direzione = 0;
      potenza = 0;
      posizione = new Point();
      numRobot++;
   }

   Robot(int d, int pot) {
      this();
      direzione = d;
      potenza = pot;
   }

   Robot(Point pos) {
      this();
      posizione = pos;
   }

   public Point avanza(int passi) {
      // calcola nuova posizione
      // sposta robot, aggiorna potenza
      return posizione;
   }
```

```
   public int carica (int c) {
      potenza = c;
      return potenza;
   }

   public boolean haiCarica() {
      return (this.potenza > POTMIN);
   }

   public int setDir (int d) {
      direzione = d%360;
      return direzione;
   }

   public int giraDestra(int gradi) {
      return setDir(direzione+gradi);
   }

   public static int getNumRobot() {
      return numRobot;
   }
```

1. Completare il codice del costruttore `Robot(int d, int pot)` usando il costruttore già definito senza parametri;
2. Completare il codice del costruttore `Robot(Point pos)` usando i costruttori già definiti;
3. Completare il codice del metodo `public boolean haiCarica()`;
4. Completare il codice del metodo `int giraDestra(int gradi)` utilizzando il metodo dato `setDir`.
5. Dichiarate e inizializzate una variabile `tartaruga`

```
Robot tartaruga = new Robot();
```

6. Date le istruzioni seguenti:

```
Point origine = new Point(100, 100);
Robot formica = new Robot(origine);
```

Aggiornate il campo `potenza` della variabile di tipo `Robot` in modo da rendere possibile il suo movimento e scrivere un'istruzione in cui si controlli che ci sia potenza sufficiente per avanzare di `n` passi usando il metodo `avanza`

```
formica.carica(150);
int passi = <leggi.intero.da.input>;
```

```
while (formica.haiCarica() && passi-->0)
    formica.avanza(1);
```

7. Considerate tutte le istruzioni eseguite fino ad ora, valutate **`Robot.getNumRobot():`** **2**

2

Data la classe **`Automobile`**:

```
public class Automobile {
   int cilindrata;
   String nome;

   Automobile() {
      cilindrata = 1000;
      nome = new String();
   }

   Automobile(int n, String p){
      this();
      this.setCilindrata(n);
      this.setNome(p);
   }
}
```

```
public String setNome(String p){
   nome = p;
   return nome;
}

public int setCilindrata(int n){
   return cilindrata = n;
}

public String getNome (){
   return nome;
}
```

1. Completate il codice del metodo **`setCilindrata`** con il prototipo **`int setCilindrata (int n).`**
2. Definite un altro costruttore con parametri utilizzando il costruttore già definito e i metodi **`setNome`** e **`setCilindrata`** per inizializzare i campi ai valori passati come parametri.
3. Infine definite il metodo **`getNome`** con il prototipo **`String getNome ()`**

3

Supponendo di aver dichiarato una variabile **`garage`**

```
Automobile[] garage = new Automobile[3];
```

Valutare il risultato dell'esecuzione delle istruzioni:

`for (int i=0; i < garage.length; i++)` `  System.out.println(garage[i]);`	`null` `null` `null`

`for (int i=0; i < garage.length; i++)`
`  System.out.println(garage[i].getNome());`
	errore
	`Exception in thread "main"`
	`java.lang.NullPointerException`

Inizializzate la variabile **`garage`** a vostro piacimento:

```
for (int i = 0; i < garage.length; i++) {
  garage[i] = new Automobile(1100, "FIAT");
```

Valutare ora il risultato dell'esecuzione delle istruzioni:

`for (int i=0; i < garage.length; i++)` `  System.out.println(garage [i].getNome());`	`FIAT` `FIAT` `FIAT`

4

Data la seguente dichiarazione con inizializzazione:

```
String[] cesto =
{"mele", "pere", "ciliege", "arance"};
```

Dire qual'è l'indice della stringa **pere**	`1`

Scrivere un ciclo per stampare tutte le occorrenze delle stringhe dell'array **cesto** con esclusione di una particolare, ad esempio **ciliege**.	`for (i=0; i<cesto.length; i++) {`
	`  if (cesto[i].equals("ciliege"))`
	`  continue;`
	`  System.out.println(cesto [i]);`
	`}`

Scrivere un'espressione che riporti la stringa **arance** conoscendone la posizione	`cesto [3]`
Generare il carattere '**e**' in **ciliege** conoscendone la posizione	`cesto [2].charAt(4);`
Scrivere un'espressione che riporti l'indice di un'occorrenza del carattere '**i**' in **ciliege**	`cesto [2].indexOf('i');`
Per conoscere la lunghezza dell'array si usa **cesto.length** o **cesto.length()**?	`cesto.length`
Scrivere un'espressione per riportare la lunghezza della stringa archiviata nella posizione **i** dell'array **cesto**	`cesto [i].length()`
Scrivere un'espressione per selezionare l'ultimo elemento dell'array **cesto** indipendentemente dal numero di elementi	`cesto [cesto.length – 1]`
Scrivere un espressione per produrre la stringa **MELE**	`cesto [0].toUpperCase()`
Dire se si è modificata la variabile **cesto**	**non è stata modificata**

5

Supponete di avere 4 variabili booleane: **gioco studio pioggia lavoro**
e l'istruzione condizionale: **if (!pioggia&&gioco) if (studio||lavoro) System.out.println("*pazienza*"); else System.out.println("*il momento giusto*"); else System.out.println("*un po' di fatica*";**
Dite per quali valori delle 4 variabili verra' stampata la frase **"*il momento giusto*"**

Si consiglia di formattare l'istruzione in modo da renderla più leggibile

```
if (!pioggia&&gioco)
   if (studio||lavoro)
      System.out.println("pazienza");
   else System.out.println("il momento giusto");
else System.out.println("un po' di fatica");
```

e di servirsi di una tabella di verità opportune:

pioggia	gioco	!pioggia	!pioggia && gioco	studio	lavoro	studio ||lavoro
F	V	V	V	F	F	F

Occorre che sia eseguito il ramo `if` della I istruzione `if-else`, cioè che la condizione **(!pioggia&&gioco)** sia vera; quindi occorre che sia eseguito il ramo `else` della II istruzione `if-else` e quindi che la condizione **(studio||lavoro)** sia falsa.

Entrambe queste condizioni si verificano in un solo caso:

pioggia = F, gioco = V, studio = F e lavoro = F

6

Assumendo le seguenti dichiarazioni iniziali

```
final int MIN= 3, LIMITE = 6;
int num1 = 1, num2 = 5, num3 = 9;
```

Scrivere l'output dei seguenti spezzoni di codice considerati separatamente e valutare il valore delle variabili all'uscita di ciascun ciclo:

`while (num1 < LIMITE) {`
`  if (num3/num1++ < MIN)`
`  System.out.println ("uno");`
`  else System.out.println ("due");`
`}`
	num1 ? 6
	num2 ? 5
	num3 ? 9
	due due due uno uno

`while (num2 >= MIN) {`
`  if (num2-- % 2 != 0) continue;`
`  System.out.println (num2);`
`}`
	num1 ? 1
	num2 ? 2
	num3 ? 9
	3

`do`
`  num2 += num3++;`
`while (num3 < LIMITE);`
	num1 ? 1
	num2 ? 14
	num3 ? 10

7

In una data classe, è definito il metodo **main** seguente che inizializza un array di parole e le stampa.

```
public static void main (String[] a) {
    final static int DIM = 5;
    String[] paroliere = archivia();
    for(int i=0; i<DIM; i++)
        System.out.println(paroliere[i]);
    System.out.println(paroliere.length);
}
```

Scrivere il prototipo del metodo statico **archivia()** che viene usato per inizializzare l'array **paroliere**:

```
public static String[] archivia()
```

Scrivere il codice del metodo **archivia()** che riporta all'ambiente chiamante un array di dimensione **DIM** (che supporremo convenientemente ampia) inizializzato con le parole lette da input. Si deve realizzare un ciclo di lettura che controlla che il numero delle parole lette non superi la dimensione massima dell'array e che termini comunque alla parola **fine**, che non viene archiviata.

```
public static String[] archivia()  {
  int count=0;
  String[] paroliere = new String[DIM];

  System.out.print("inserisci nomi, \"fine\" per terminare: ");
  String input;

  while (count<DIM && !(input=Keyboard.readString()).equals("fine"))
    paroliere[count++]=input;

  return paroliere;
}
```

Scrivere l'output dell'esecuzione del metodo **main** nell'ipotesi di input:

casa cane dado fine

```
casa
cane
dado
null
null
5
```