Esame di Programmazione II, 21 giugno 2018

Si consideri un'interfaccia che rappresenta un numero non negativo, in una qualsiasi base di numerazione:

```
public interface Number {
  int getValue();
}
```

Esercizio 1 [8 punti] Si completi la seguente implementazione astratta di un `Number`, che fornisce le funzionalità comuni a tutti i numeri, cioè il controllo sulla non negatività del valore, l'accesso al valore, la traduzione in stringa e i metodi per il test di uguaglianza:

```
public abstract class AbstractNumber implements Number {
  private final int value;

  protected AbstractNumber(int value) {
    // lancia una IllegalArgumentException se value e' negativo,
    // altrimenti inizializza il campo value
    ...
  }

  // restituisce il valore di questo numero
  @Override public final int getValue() { ... }

  // restituice la base di numerazione di questo numero
  protected abstract int getBase();

  // restituisce il carattere che rappresenta la cifra "digit" nella base di numerazione
  // di questo numero. Sara' sempre vero che 0 <= digit < getBase()
  protected abstract char getCharForDigit(int digit);

  // restituisce una stringa che rappresenta il numero nella sua base di numerazione
  @Override public String toString() { ... }

  @Override public final boolean equals(Object other) {
    // due numeri sono uguali se e solo se hanno lo stesso valore
    ...
  }

  @Override public final int hashCode() {
    // deve essere compatibile con equals() e non banale
    ...
  }
}
```

Esercizio 2 [6 punti] Si scrivano le sottoclassi concrete `DecimalNumber`, `BinaryNumber`, `OctalNumber` e `HexNumber` di `AbstractNumber`, che rappresentano, rispettivamente, un numero in base 10, 2, 8 e 16. Queste classi si instanziano con il loro costruttore, a cui viene passato il valore del numero. Non si ridefinisca, in queste quattro sottoclassi, il metodo `toString()`: quello ereditato da `AbstractNumber` dovrà funzionare per tutte queste sottoclassi, traducendo il valore del numero nella giusta base di numerazione.

Esercizio 3 [**5 punti**] Nella codifica binaria con parità, un numero binario viene esteso con un'ulteriore cifra binaria di controllo, in modo da rendere pari il numero totale di cifre `1`. Se quindi il numero binario aveva una quantità pari di `1`, si aggiungerà una cifra di controllo `0`. Se invece il numero binario aveva una quantità dispari di `1`, si aggiungerà una cifra di controllo `1`. Questa modifica riduce il rischio di trasmissione di dati corrotti, permettendo di implementare un rudimentale sistema di rilevazione dell'errore. Si implementi una sottoclasse concreta `BinaryNumberWithParity` di `BinaryNumber`, ridefinendo il metodo `toString()` in modo da aggiungere in fondo la cifra di controllo opportuna.

Esercizio 4 [**5 punti**] Nella codifica in base 58, si utilizzano 58 cifre diverse, scelte fra i numeri arabi e le lettere inglesi maiuscole e minuscole. Si evitano i caratteri `OOIl`, che potrebbero essere confusi a video, perché graficamente simili. Si implementi una sottoclasse concreta `Base58Number` di `AbstractNumber`, in modo da implementare questa numerazione in base 58. Le 58 cifre sono quindi `123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz`. Non si ridefinisca il metodo `toString()` ereditato da `AbstractNumber`.

Esercizio 5 [**5 punti**] Si completi la seguente classe di prova:

```
public class Main {
  public static void main(String[] args) {
    try (Scanner keyboard = new Scanner(System.in)) {
      System.out.print("Inserisci un numero non negativo: ");
      int n = keyboard.nextInt();

      List<Number> numbersAsList = new ArrayList<>();
      // si aggiunga a numbersAsList il numero n, 6 volte:
      // prima in base 10, poi in base 2, poi 2 con parita', poi 8, poi 16 e infine 58
      ...
      System.out.println("lista: " + numbersAsList);

      Set<Number> numbersAsSet = ...
      // si copino tutti gli elementi della lista numbersAsList dentro l'insieme numbersAsSet
      ...
      System.out.println("insieme: " + numbersAsSet);
    }
    // se si verifica una IllegalArgumentException, la si intercetti, stampando su video
    // un messaggio che chiarisca che ci si aspettava un numero non negativo
    ...
  }
}
```

Se tutto è corretto, l'esecuzione della classe `Main` dovrà stampare ad esempio:

```
Inserisci un numero non negativo: 1234567
lista: [1234567, 100101101011010000111, 1001011010110100001111, 4553207, 12D687, 7Kze]
insieme: ...
```

Esercizio 6 [**2 punti**] Cosa verrà stampato alla linea precedente, al posto dei puntini di sospensione?

Si possono definire altri campi, metodi, costruttori e classi, ma solo `private`.