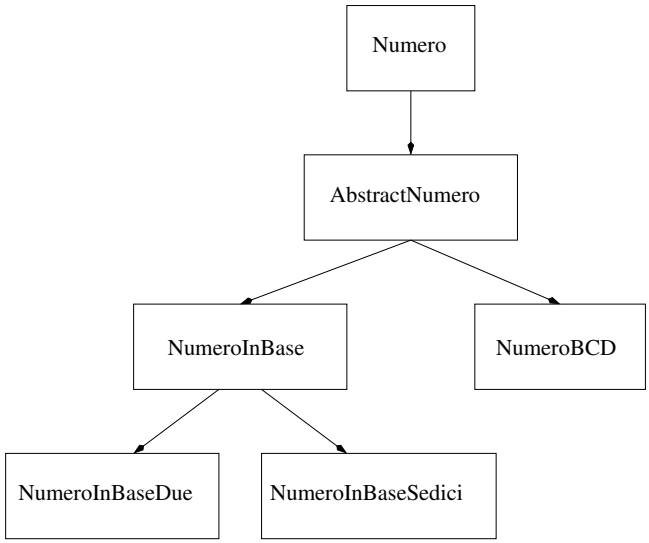
Esame di Programmazione II, 27 settembre 2013 (2 ore)

Si consideri una gerarchia di classi che rappresentano dei numeri interi:



L'interfaccia `Numero` è definita come:

```
public interface Numero extends Comparable<Numero> {
  public int getValue();
  public void aggiungi(Numero n);
  public void sottrai(Numero n);
}
```

dove il metodo `getValue` restituisce il valore intero del numero e i metodi `aggiungi` e `sottrai` modificano il numero aggiungendo o sottraendone un altro, rispettivamente. Se la sottrazione desse origine a un numero negativo, dovrebbe venire lanciata un `java.lang.ArithmeticException`. La comparazione fra due numeri è il loro ordinamento rispetto al valore crescente. Due numeri sono uguali se e solo se hanno lo stesso valore.

Esercizio 1 [5 punti] Si scriva la classe astratta `AbstractNumero` che implementa, come `final`, i metodi `getValue`, `aggiungi`, `sottrai`, `compareTo`, `equals` e `hashCode`. Se servono campi, devono essere dichiarati `private`. Se servono costruttori, devono essere dichiarati `protected`.

Esercizio 2 [1 punti] Si scriva la classe astratta `NumeroInBase` che implementa un numero in una base di numerazione tra due e sedici. Deve fornire un metodo `final` `getBase` che restituisce la base di numerazione del numero. Deve avere un costruttore

```
protected NumeroInBase(int value, int base)
```

che costruisce il numero `value` nella base indicata e che lancia una `java.lang.ArithmeticException` se la base non è fra due e sedici.

Esercizio 3 [4 punti] Si scriva la classe concreta `NumeroInBaseDue` che implementa un numero in base due (la normale codifica binaria dei numeri interi). Deve avere un costruttore

```
public NumeroInBaseDue(int value)
```

che costruisce il numero `value` in base due e che lancia una `java.lang.ArithmeticException` se il valore è negativo. Deve avere un metodo `toString` che restituisce una stringa che descrive il numero (quindi fatta solo di `'0'` e `'1'`).

Esercizio 4 [4 punti] Si scriva la classe concreta `NumeroInBaseSedici` che implementa un numero in base sedici (la normale codifica esadecimale dei numeri interi). Deve avere un costruttore

```
public NumeroInBaseSedici(int value)
```

che costruisce il numero `value` in base sedici e che lancia una `java.lang.ArithmeticException` se il valore è negativo. Deve avere un metodo `toString` che restituisce una stringa che descrive il numero in esadecimale.

Esercizio 5 [6 punti] La rappresentazione *binaria a codice decimale* (bcd) è un modo di scrivere in binario i numeri decimali non negativi, in cui si riservano quattro cifre binarie per ogni cifra decimale. Per esempio, il numero decimale 209 viene scritto in bcd come:

$$\underbrace{0010}_{2}\,\underbrace{0000}_{0}\,\underbrace{1001}_{9}$$

Si scriva la classe concreta `NumeroBCD` che implementa un numero in binario a codice decimale. Deve avere un costruttore

```
public NumeroBCD(int value)
```

che costruisce il numero `value` in binario a codice decimale e che lancia una `java.lang.ArithmeticException` se il valore è negativo. Deve avere un metodo `toString` che restituisce una stringa che descrive il numero in binario a codice decimale (quindi fatta solo di `'0'` e `'1'`).

* * * * *

Se tutto è corretto, l'esecuzione del seguente programma:

```
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

public class Main {
  public static void main(String[] args) {
    Numero n1 = new NumeroInBaseDue(2034);
    Numero n2 = new NumeroInBaseSedici(2034);
    Numero n3 = new NumeroBCD(2034);
    System.out.println("n1=" + n1);
    System.out.println("n2=" + n2);
    System.out.println("n3=" + n3);
    n2.aggiungi(n3);
    n2.aggiungi(n1);
    n2.sottrai(new NumeroBCD(136));
    System.out.println("n2=" + n2);

    Map<Numero, String> map = new HashMap<Numero, String>();
    map.put(n1, "duemilatrentaquattro in base due");
    map.put(n3, "duemilatrentaquattro in binario a codice decimale");
    System.out.println(map.get(n1)); // cosa stampa?

    // java.util.TreeSet e' un insieme ordinato in senso crescente rispetto a compareTo
    Set<Numero> insieme = new TreeSet<Numero>();
    insieme.add(n1);
    insieme.add(n2);
    insieme.add(n3);
    System.out.println(insieme); // cosa stampa?
  }
}
```

deve stampare:

```
n1=11111110010
n2=7f2
n3=0010000000110100
n2=174e
.......................
.......................
```

Esercizio 6 [2 punti] Cosa viene stampato (al posto dei puntini) dalle ultime due istruzioni `System.out.println` del metodo `main` della classe `Main`? Perché?