Esame di Programmazione II, 23 febbraio 2015

Il gioco della vita è una tabella bidimensionale di cellule, le quali possono essere vive (rappresentante con un asterisco) oppure morte (rappresentate con uno spazio). Per esempio, la seguente è una rappresentazione di un gioco della vita di dimensione 40×7, in un certo istante:

```
  **
  **                    **
                        * *
                        *

     ***
```

È possibile cambiare la tabella di cellule secondo due regole:

1. una cellula morta con esattamente 3 vicini vivi viene sostituita con una cellula viva;
2. una cellula viva con 2 o 3 vicini vivi viene sostituita con una cellula viva; altrimenti viene sostituita con una cellula morta (per isolamento o sovraffollamento).

Esercizio 1 [6 punti] Si completi la seguente classe, che implementa una cellula.

```
public class Cell {
  private final boolean alive; // viva o morta?
  public Cell(boolean alive) { ... } // costruisce una cellula, specificando se viva o morta
  public String toString() { ... } // stringa di un solo carattere, asterisco o spazio
  public Cell next(Iterable<Cell> neighbors) { ... }
}
```

Il metodo `next` restituisce la cellula che deve sostituire `this`, implementando le due regole del gioco date sopra. Il parametro `neighbors` sono i vicini di `this`, la cui vita o morte determina la scelta della cellula che deve sostituire `this`.

Esercizio 2 [5 punti] Nel gioco della vita esistono delle *figure*, cioè delle combinazioni di cellule che prendono nomi ben precisi. Per esempio, un *blocco* è una figura fatta da quattro cellule vive:

```
**
**
```

La seguente classe implementa una figura, che conosce le coordinate di partenza, in alto a sinistra, in cui la figura è posizionata dentro una tabella di cellule (le coordinate crescono verso il basso e verso destra):

```
public abstract class Figure {
  protected final int startX;
  protected final int startY;

  protected Figure(int startX, int startY) {
    this.startX = startX;
    this.startY = startY;
  }

  public abstract Cell mkAliveCellAt(int x, int y);
}
```

Il metodo `mkAliveCellAt` restituisce la cellula alle coordinate x e y della tabella di cellule solo se appartiene alla figura ed è viva; altrimenti ritorna `null`. Per esempio, la seguente è l'implementazione di un blocco:

```
public class Block extends Figure {
  public Block(int startX, int startY) {
    super(startX, startY);
  }

  @Override
  public Cell mkAliveCellAt(int x, int y) {
    x = x - startX; // calcola lo spostamento rispetto all'angolo in alto a sinistra della figura
    y = y - startY;

    // le cellule vive sono in coordinate tra 0 e 1, sia ascissa che ordinata
    return 0 <= x && x <= 1 && 0 <= y && y <= 1 ? new Cell(true) : null;
  }
}
```

Si scrivano altre sottoclassi di `Figura`, in modo da implementare:

- un *lampeggiatore* di dimensione 3×1 (classe `Blinker.java`):

  ```
  ***
  ```

- una *barca* di dimensione 3×3 (classe `Ship.java`):

  ```
  **
  * *
   *
  ```

- un *rospo* di dimensione 4×2 (classe `Toad.java`):

  ```
   ***
  ***
  ```

- un *aliante* di dimensione 3×3 (classe `Glider.java`):

  ```
  ***
  *
   *
  ```

Esercizio 3 [**11 punti**] La tabella di cellule contiene le cellule del gioco e permette di sostituirle con quelle che risultano dalla due regole viste sopra. Si completi la seguente classe che implementa la tabella:

```
public class Board {
  private final int width;
  private final int height;
  private final Cell[][] cells;

  // costruisce una tabella delle dimensioni indicate e in cui sono posizionate
  // le figure indicate. Altrove le cellule devono essere inizialmente morte
  public Board(int width, int height, Figure... figures) { ... }

  // fa passare le cellule della tabella alla loro configurazione successiva
  public void recomputeCells() { ... }

  // restituisce una descrizione della tabella, fatta da spazi e asterischi,
  // come quella mostrata all'inizio del compito
  public String toString() { ... }
}
```