SubHunter Refactored

This is my simple refactoring of SubHunter as discussed in class. It's not meant to represent a definitive refactoring, it is simply one set of choices for refactoring. Some details have been removed to keep the document size manageable.

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
class URandomK{
   public URandomK(int k){ . . .}
   public int nextInt(){ . . .  }          // returns next unused integer less than K
}
```

Abstract Grid Cell provides an example of a state pattern

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// AbstractGridCell represents a single grid cell in the game grid. It is the base class for
// each of the different state classes that each grid cell can represent.
//
abstract class AbstractGridCell{
   private int x,y,w,h;

   //``````````````````````````````````````````````````````````````````````````````
   public AbstractGridCell(int x, int y, int w, int h){this.x=x; this.y=y; this.w=w; this.h=h;}
   public AbstractGridCell(AbstractGridCell myCell){this(myCell.x, myCell.y, myCell.w, myCell.h);}

   //``````````````````````````````````````````````````````````````````````````````
   public void drawGrid(Canvas canvas, Paint paint, int fillColor){
        . . .   //Here you draw the grid with the known size given in x,y,w,h
   }

   //``````````````````````````````````````````````````````````````````````````````
   public void drawGrid(Canvas canvas, Paint paint)    {drawGrid(canvas, paint, Color.WHITE);}
   public AbstractGridCell takeShot()                  {return this;}
   public AbstractGridCell clearShot()                 {return this;}
}
```

These classes extend the abstract state to represent the state for each cell.

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
class EmptyGridCell extends AbstractGridCell{
   public EmptyGridCell(int x, int y, int w, int h)    {super(x,y,w,h);}
   public EmptyGridCell(AbstractGridCell myCell)       {super(myCell);}
   public AbstractGridCell takeShot()                  {return new Shot(this);}
};

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
class Sub extends AbstractGridCell{
   public Sub(AbstractGridCell myCell)                 {super(myCell);}
   public AbstractGridCell takeShot()                  {return new SunkSub(this);}
};

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
class SunkSub extends AbstractGridCell{
   SunkSub(AbstractGridCell myCell)                    {super(myCell);}
   public void drawGrid(Canvas canvas, Paint paint)    {super.drawGrid(canvas,paint,Color.RED);}
}
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
class Shot extends AbstractGridCell{
   Shot(AbstractGridCell myCell){super(myCell);}
   public void drawGrid(Canvas canvas, Paint paint)    {super.drawGrid(canvas,paint,Color.BLUE);}
   public AbstractGridCell clearShot()                 {return new EmptyGridCell(this);}
}
```

The Grid class on the next two pages is the heart of the application and manages a collection of grid cells and their interaction.

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
class Grid{
   private class GridPosition{
       int x,y;
       public GridPosition(int x, int y) {this.x=x; this.y=y;}
   }
       . . . size of grid is comprised of private variables in this class, see code

   private Vector<AbstractGridCell> gridCells;
   private List<Sub> subs;

   //```````````````````````````````````````````````````````````````````````````````````````
   public Grid(int x, int y){
       . . . // Setup the private class size variables
       reset();
   }

   //```````````````````````````````````````````````````````````````````````````````````````
   public void reset(){
       rand = new URandomK(gridHeight*gridWidth);
       gridCells = new Vector<AbstractGridCell>(gridHeight*gridWidth);

       for(int h=0; h<gridWidth; h++)
           for(int v=0; v<gridHeight; v++)
               gridCells.add(new EmptyGridCell(h*blockSize,v*blockSize,blockSize,blockSize));
   }

   //```````````````````````````````````````````````````````````````````````````````````````
   public void spawnNewSub(){
       int subCell = rand.nextInt();
       gridCells.set(subCell, new Sub(gridCells.get(subCell)));
   }

  //````````````````````````````````````````````````````````````````````````````````````````
   public int sunkSubCount(){
       int sunkSubs=0;
       for(AbstractGridCell agc:gridCells) {
           if(agc instanceof SunkSub) // ← note the use of instanceof to determine object type
               sunkSubs++;
       }
       return sunkSubs;
   }
```

```java
//``````````````````````````````````````````````````````````````````````````````````
   private int gridCellN(GridPosition p){return (gridHeight*p.x+p.y);}

   //``````````````````````````````````````````````````````````````````````````````
   private int distanceToClosestSubFrom(GridPosition shotP){
      int subD=gridWidth*gridHeight;

      for(int i=0; i< gridCells.size(); i++) {
         AbstractGridCell agc = gridCells.get(i);
         if(agc instanceof Sub){
              // set subD to existing min, or distance from agc to shotP
              . . .
         }
      }
      return subD;
   }

   //``````````````````````````````````````````````````````````````````````````````
   public int takeShot(float touchX, float touchY){
      GridPosition tP = new GridPosition( (int)touchX/ blockSize,
              (int)touchY/ blockSize);

       // Note that the essence of changing state happens here
       //
      gridCells.set(lastShot,gridCells.get(lastShot).clearShot());
      gridCells.set(gridCellN(tP),gridCells.get(gridCellN(tP)).takeShot());
      lastShot=gridCellN(tP);
      return distanceToClosestSubFrom(tP);
   }

   //``````````````````````````````````````````````````````````````````````````````
   public void drawGrid(Canvas canvas, Paint paint){
      for(AbstractGridCell agc:gridCells)
         agc.drawGrid(canvas,paint);
   }

   //``````````````````````````````````````````````````````````````````````````````
   public int getBlockSize(){return blockSize;}
}
```

Finally, the game and game activity classes manage the grid. Note, much of the code has been removed as it does not apply to refactoring.

```java
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
class SubHunterGame{
   static final int numSubs = 4;
       . . .   // essential local vars go here including counts and bitmap et al.
   Grid grid;

   public SubHunterGame(Context context, Point size){
     // canvas setup goes here
      grid = new Grid(size.x, size.y);
      newGame();
   }

   //``````````````````````````````````````````````````````````````````````````````
```

```java
    void newGame(){
        . . . // reset game and grid

        for(int i=0; i<numSubs; i++)
            grid.spawnNewSub();
        draw();
    }

    //``````````````````````````````````````````````````````````````````````````````````
    void draw() {
        . . . draw grid and any text
    }

    //``````````````````````````````````````````````````````````````````````````````````
    void takeShot(float touchX, float touchY){
        distanceFromSub = grid.takeShot(touchX, touchY);
        shotsTaken++;
        if(resetGame)                           {newGame();}
        else if(grid.sunkSubCount()>=numSubs)   {boom();}
        else                                    {draw();}
    }

    //``````````````````````````````````````````````````````````````````````````````````
    void boom(){ . . .}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
public class SubHunterRFActivity extends Activity {
    SubHunterGame game;

    protected void onCreate(Bundle savedInstanceState) {
        . . . // setup game
    }

    public boolean onTouchEvent(MotionEvent motionEvent) {
        if((motionEvent.getAction() & MotionEvent.ACTION_MASK) == MotionEvent.ACTION_UP) {
            game.takeShot(motionEvent.getX(), motionEvent.getY());
        }
        return true;
    }
}
```