贪食蛇 Java 详细教程

首先来产生一个窗口，这将是之后绘制所有东西的容器，这里使用 awt 类库的 Frame 类来产生一个窗口，新建一个文件 Board.java，加入如下代码

```
import java.awt.*;
import java.awt.event.*;

class Board extends Frame
{
}
```

让这个类从 Frame 类继承而来，类与类之间一共有两种关系，继承与包含，继承的含义是，什么是一个什么，比如这里的 Board 是一个 Frame，而包含的含义是什么有一个什么，所以在选择继承与包含时一定要先考虑清楚这个问题。PS： 大多数情况下用的都是包含关系。

完成类之后就要写构造方法，构造方法最重要的含义就是如何创建这个对象，它是创建对象时被自动调用的，没有返回值，因为构造函数返回的都是一个新对象。

```
public  Board(String title)
{
        super(title);
        setSize(400, 300);
        setVisible(true);
        setResizable(false);
}
```

这里有一句很重要的名言，构造子类之前一定要先构造父类，而且一定是最先构造父类，这里调用父类的构造函数来完成对父类的构造，紧跟着调用 setSize、setVisible、setResizable，setSize 设置窗口大小，为什么要调用 setVisible 呢？还有 setResizable 又是干什么的？别想了，你就是那 95%没有打开 API 手册的人，如果还没有一份 API 参考的话，请停止下面的阅读，因为下面的很多东西都是信赖它的。

这是关于 Frame 构造函数的说明，这里我们传一个字符串，所以调用的是第三个构造函数，注意下面的备注，“最初不可见”，那么我们的窗口肯定是要它可见的，所以调用 setVisible，这个函数可以在 Frame 的父类 Window 中找到，同理 setResizable 也能找到相关说明，它将窗口大小固定，因为我们不希望我们的玩家改变窗口大小。

构造方法摘要

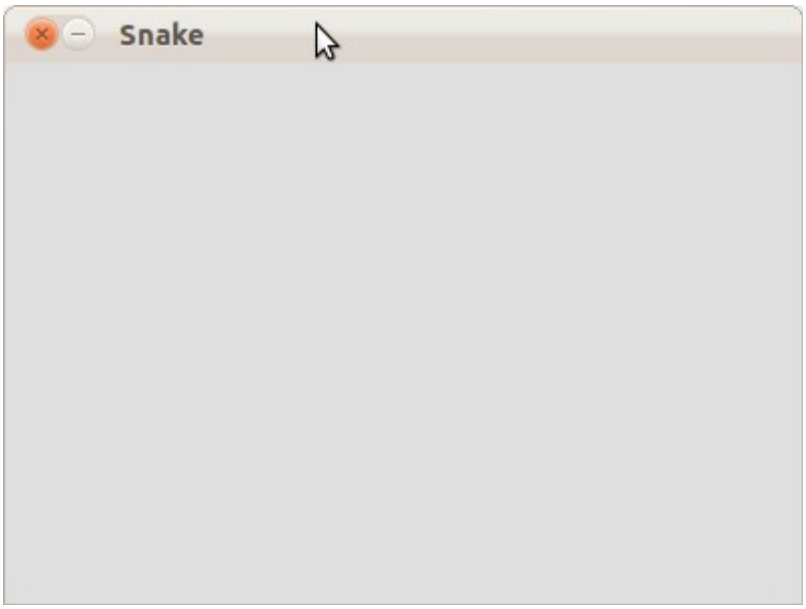
Frame() 构造一个最初不可见的 Frame 新实例 () 。
Frame(GraphicsConfiguration gc) 使用指定的 GraphicsConfiguration 构造一个最初不可见的新 Frame。
Frame(String title) 构造一个新的、最初不可见的、具有指定标题的 Frame 对象。
Frame(String title, GraphicsConfiguration gc) 构造一个新的、初始不可见的、具有指定标题和 GraphicsConfiguration 的 Frame 对象。

至于为什么将窗口大小设为 400，300，这是有原因的，为将蛇的每一节设为 16 像素，游戏盘为 20x15 的，那么最小需要 360*240 的大小，然后四周再留一些空余，一为美观，二可以添加一些信息。

接下来在写一个 main 函数

```
public static void main(String args[])
{
        Board board = new Board(“Snake”);
}
```

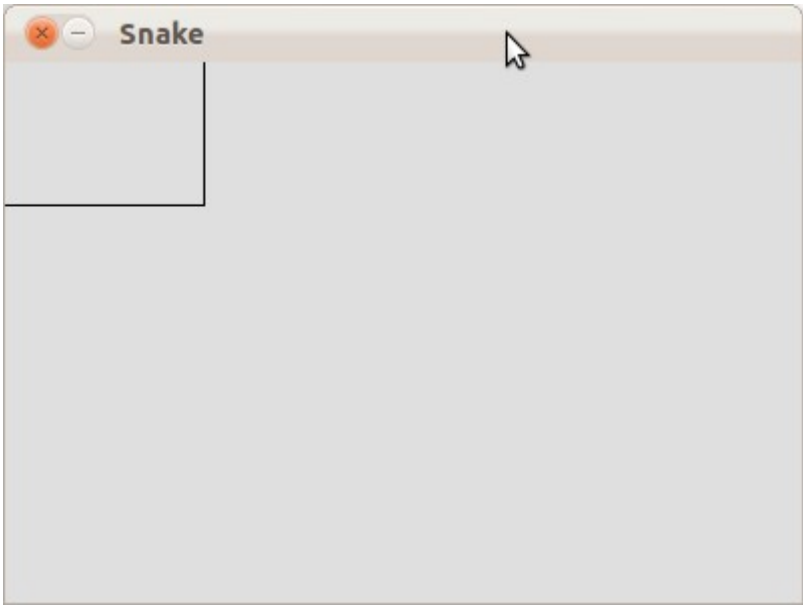
编译运行可看到如下结果，你会发现它还不能被关闭，这点以后解决，目前你可以在命令行中按 ctrl+C 来结束它



现在来往里面画点东西，要画东西，有一个 paint 函数，它传了一个 Graphics 对象，这个 Graphics 可以理解为画笔，当窗口被创建的时候会自动调用 paint 函数来完成图像的绘制。

继续加入下列语句

```
public void paint(Graphics g)
{
        g.drawRect(0, 0, 100, 100);
}
```



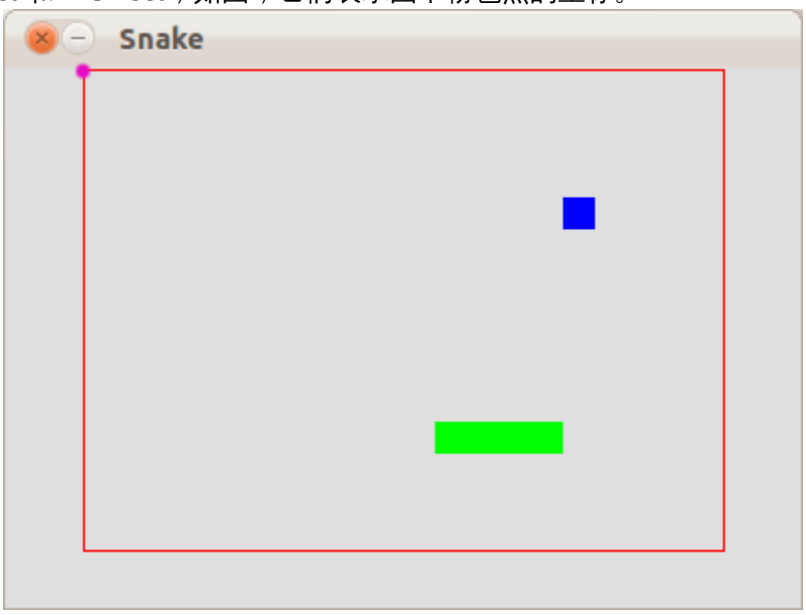
你可能会发现这个矩形并没有显示全，它有一部分被标题栏挡住了，所以这里有一个性质：画图都是从窗口左上角开始画的。

好了，上面都是一个基本的测试工作，下面要来真正设计我们的 Board 类了，首先在类中定义一些常量，直接出现 400，300，16 之类的数（这种数也叫魔数）是很危险的，一旦某个值发生改变，则用到这个数的地方都要变，如果把它们定义成常量，需要用到这个数的时候我们引用这些常量，这样当这个常量发生变化时我们只要修改常量的值就可以了。

在 Board 类中加入下列成员变量

```
private static final int Width = 400;                    //窗口的宽度与高度
private static final int Height = 300;
private static final int TileWidth = 16;                 //每一格的宽度与高度，Tile 本意指瓷砖，
private static final int TileHeight = 16;                //在游戏在专门指每一个单元
public  static final int Row = 15;                       //游戏盘的行数与列数
public  static final int Column = 20;
private static final int XOffset = (Width - Column*TileWidth)/2;  //当我们确定游戏盘大小后，
private static final int YOffset = (Height- Row*TileHeight)/2; //也就是蛇运动的区域，这个左上
```

角的坐标叫 XOffset 和 YOffset，如图，它们表示图中粉色点的坐标。



然后把构造函数写成如下形式

```
public Board(String title)
{
        super(title);
        setSize(Width, Height);
        setVisible(true);
        setResizable(false);
}
```

然后来看看一个游戏盘中有些什么，首先能想到的，一条蛇和一块食物，那就先来看蛇，把

Board.java 保存，新建一个 Snake.java，

```
class Snake
{

}
```

一条蛇又有些什么呢？一节一节的身体，那又如何表示它们呢？蛇的运动，如果你仔细观察的话，可以看出，实际上就是每次把尾巴上的一节移到脑袋前方，很自然的想到了链表，第次把链表尾部的结点删除，再向链表头插入一个新节点，就完成了一次运动。

在 Snake 中定义一个链表

```
private List<Body> mBody;
```

根据包含表示什么有什么的原则，这句代码就可以翻译成一条蛇有一个串由 Body（身体节）组成的链，逻辑上是说的通的。

在 Java 中，一切都是对象，一节一节的身体应该也是，Body 类很简单，它拥有的只是一给坐标，因为只要有了坐标你就能在游戏盘中把它画出来。保存 Snake.java，再新建一个 Body.java

```
class Body
{
	public Body(int row, int col)
	{
		this.row = row;
		this.col = col;
	}
	public int row;
	public int col;
}
```

应该很简单明了，值得一提的是，这里为什么把 row 与 col 设为 public，这是经过考虑的，应为 Body 几乎没有表现出类的特性，它更像一种数据类型，跟 int double 一样，只不过它是二元的。

让我们回到 Snake.java，对于链表，不熟悉的话我再介绍一下。

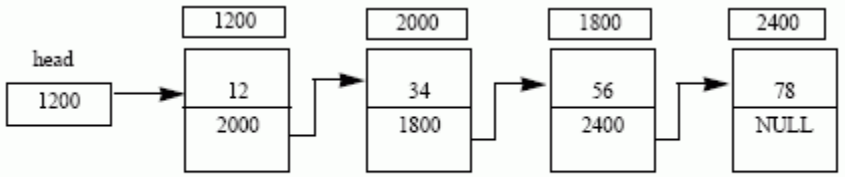


图7-3 单链表

链表由由多个结点串联而成，每个结点包含两部分，一部分是数据域，另一部分是下一结点的位置。而 Java 的 List 定义中 List<Body> 其中 Body 就是数据域的类型，这是一种复合类型，当然你也可以用简单类型如 int、float 之类的。但这个 List 只是一种接口，不理解什么是接口的话参看教材，这里不用重点不在接口，只说一点，接口是不能被实例化的，这就是说你不能写 List<Body> list = new List<Body>();

但是凡是实现了 List 这一接口的类都能实例化一个 List，这里我先择了 ArrayList;

你可以这样写 List<Body> list = new ArrayList<Body>();

～！现在别写，一会儿我们将在 Snake 的构造函数里实例化这一链表。

打开 API 手册，搜索 List，并查看，出现下页图所示。

方法摘要	
boolean	**add**(E e) 向列表的尾部添加指定的元素（可选操作）。
void	**add**(int index, E element) 在列表的指定位置插入指定元素（可选操作）。
boolean	**addAll**(Collection<? extends E> c) 添加指定 collection 中的所有元素到此列表的结
boolean	**addAll**(int index, Collection<? extends 将指定 collection 中的所有元素都插入到列表中
void	**clear**() 从列表中移除所有元素（可选操作）。
boolean	**contains**(Object o) 如果列表包含指定的元素，则返回 true。
boolean	**containsAll**(Collection<?> c) 如果列表包含指定 collection 的所有元素，则返
boolean	**equals**(Object o) 比较指定的对象与列表是否相等。
E	**get**(int index) 返回列表中指定位置的元素。
int	**hashCode**() 返回列表的哈希码值。
int	**indexOf**(Object o) 返回此列表中第一次出现的指定元素的索引；如
boolean	**isEmpty**() 如果列表不包含元素，则返回 true。
Iterator<E>	**iterator**() 返回按适当顺序在列表的元素上进行迭代的迭代
int	**lastIndexOf**(Object o) 返回此列表中最后出现的指定元素的索引；如果

这里主要用其中几个方法，整理如下：

boolean

add(E e)

向列表的尾部添加指定的元素（可选操作）。

void
add(int index, E element)
在列表的指定位置插入指定元素（可选操作）

void
clear()
从列表中移除所有元素（可选操作）。

E
get(int index)
返回列表中指定位置的元素。

E
remove(int index)
移除列表中指定位置的元素（可选操作）。

int
size()
返回列表中的元素数

注：这里的 E 在本例中就是 Body 类型。

上面只是一个简介，如果你想了解这些函数的详细说明，参见 API 手册。

继续回到 Snake.java，仔细想想 Snake 还有些什么东西，在上文提到蛇的运动实际上就是每次把尾巴上的一节移到脑袋前方，没错，方向！方向有四个，那用什么类型表示呢？你当然可以用 int 0-3 分别表示上下左右，但这是 C 语言中的做法，做为一门面向对象的语言，这种表示状态的量应选用枚举型变量，关于枚举参看教材 165 页，它的本质就是用一个单词代替一个数值，还有，注意，枚举是在类之外声明的。

在 Snake.java 的最前端加上一个枚举的声明

```
enum DIRECTION
{
    UP, DOWN, LEFT, RIGHT;
}
```

回到 Snake 类中，现在来定义方向变量

```
private DIRECTION mDirection;
```

然后再来完成 Snake 的构造函数。

```
    public Snake()
    {
        mBody = new ArrayList<Body>();
        mBody.add(new Body(7, 4));
        mBody.add(new Body(7, 3));
        mBody.add(new Body(7, 2));
```

```
            mBody.add(new Body(7, 1));
            mDirection = DIRECTION.RIGHT;
        }
```

第一行之前说过，我们用一个 ArrayList 来实例化一个 List，再调用这个 List 的 add 方法，依次添加四个身体节到这条蛇中，(7,4) (7,3)表示的是蛇身的坐标，add 将新节点挂下尾部，所以这样这条蛇在被构造出来的时候是朝右的，长度为 4。

到这，来整理一下 Snake.java 里的代码，不要遗漏些什么。

```
import java.util.List;              //不要忘了引入类文件
import java.util.ArrayList;
enum DIRECTION
{
        UP, DOWN, LEFT, RIGHT;
}
class Snake
{
        public Snake()
        {
            mBody = new ArrayList<Body>();
            mBody.add(new Body(7, 4));
            mBody.add(new Body(7, 3));
            mBody.add(new Body(7, 2));
            mBody.add(new Body(7, 1));
            mDirection = DIRECTION.RIGHT;
        }
        private List<Body> mBody;
        private DIRECTION mDirection;
}
```

接下来，看看我们需要一条蛇做哪些事情。首先是移动（Move），然后是吃食(Eat)，转向(Turn)。可能你只能想到这么多，但在后面的代码中你会发现它还得干一件事——重置。道理很简单，你的蛇挂了以后你不可能让它以原来的长度以坐标继续开始新的游戏。

先说吃食，这里吃食是不好描述的，因为整个游戏并不关心你的蛇是如何享用美餐的，而只关心吃完会怎么样（变长）。所以我把吃归结到了移动里。再看移动，Move 这个函数需要哪些参数呢？我一开始以为并不要任何参数，因为蛇的运动跟其它东西并没有关系，但很快证明是错的，蛇的运动需要了解游戏盘，以判断是否吃到了食物，是否碰到了墙壁。在 Snake 类中加入函数 move

```
public void move(Board b)
{
}
```

先留空，一会儿补全。再来看转向，很简单，一行代码，

```
public void setNextDirection(DIRECTION d)
{
	mDirection = d;
}
```

但对吗？姑且这样写，后面再说。

最后来看重置 reset，它把蛇弄回初始状态

```
	public void reset()
	{
		mBody.clear();		//清空链表先
		mBody.add(new Body(7, 4));
		mBody.add(new Body(7, 3));
		mBody.add(new Body(7, 2));
		mBody.add(new Body(7, 1));
		mDirection = DIRECTION.RIGHT;
	}
```

没错，这个函数中大多数的语句你都见过，就在 Snake 的构造函数里，事实上，对于这种代码的重复，我们应该把公共部分写在一个函数里供其它函数调用。那么构造函数可进一步改进成

```
	public Snake()
	{
		mBody = new ArrayList<Body>();
		reset();
	}
```

此外，再定义两个 get 方法 getLength（用来获取蛇的长度）和 getBody(int index)（获取蛇的某一节），为什么呢？如果你现在不定义的话，写到后面发现缺少条件，还是要回来的再写的，写这两个方法是必然的。

```
	public Body getBody(int index)
	{
		return mBody.get(index);
	}
	public int getLength()
	{
		return mBody.size();
	}
```

get 与 size 的用途上面已经列出，这两个函数的意义显而易见。

最终你的代码是这个样子，请确保 Snake.java 能通过编译，然后继续后面的过程。

```
import java.util.List;
import java.util.ArrayList;

enum DIRECTION
{
	UP, DOWN, LEFT, RIGHT;
}
class Snake
{
```

```
	public Snake()
	{
		mBody = new ArrayList<Body>();
		reset();
	}
	private List<Body> mBody;
	private DIRECTION mDirection;
	public void move(Board b)
	{
	}
	public void setDirection(DIRECTION d)
	{
		mDirection = d;
	}
	public Body getBody(int index)
	{
		return mBody.get(index);
	}

	public int getLength()
	{
		return mBody.size();
	}
	public void reset()
	{
		mBody.clear();
		mBody.add(new Body(7, 4));
		mBody.add(new Body(7, 3));
		mBody.add(new Body(7, 2));
		mBody.add(new Body(7, 1));
		mDirection = DIRECTION.RIGHT;
	}
}
```

保存 Snake 类，我想信你应该有点感到厌烦了，再这样也许你会崩溃的，现在来做一点有意思的事情。

打开 Board.java 文件，清空 paint 里的内容，（如果你没清空的话），一样的，来看看需要我们的游戏盘完成哪些任务。。。。好吧，画蛇。但在画蛇之前，请允许我先把蛇的运动区域画出来。

在 Board 类中添加一个函数 drawDecoration，之所以不叫 drawBounds，是因为你以后可以添加一些美化，比如放一个显示分数的东西。

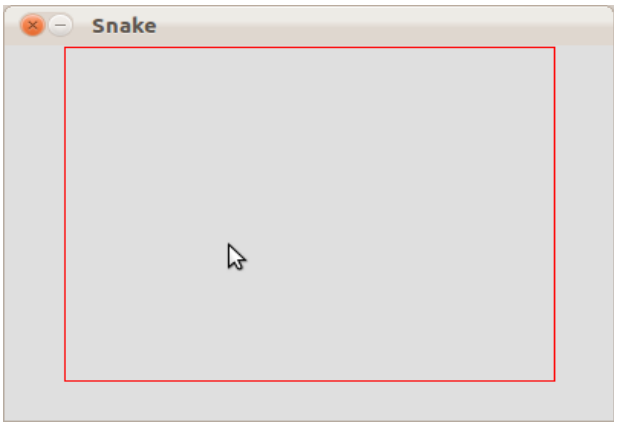
```
	public void drawDecoration(Graphics g)
	{
		g.setColor(Color.RED);
		g.drawRect(XOffset, YOffset, Column*TileWidth, Row*TileHeight);
	}
```

然后再在 paint 函数里调用 drawDecoration

```
	public void paint(Graphics g)
```

```
    {
        drawDecoration(g);
    }
```

如果你现在运行你可以看到一个红色的红框。



如果你要画一条蛇，至少得有一条蛇吧，注意这里的 Board 包含了整个游戏盘，而不仅仅是一个窗口，意思 paint 的职责是绘制属于游戏盘中第一个可见的东西，否则就是“没尽职”。同样的，如果它绘制了一个不属于游戏盘中的东西，那就是多管闲事了。由以上两条，可以得出，如果要让游戏盘显示蛇，那就必须要让蛇成为游戏盘的一部分，简而言之：private Snake mSnake;

```
    在 Board 的构造函数中实例化这条蛇
    mSnake = new Snake();
```

再添加一个 drawSnake 函数用来绘制这条蛇

```
    public void drawSnake(Graphics g)
    {
        g.setColor(Color.GREEN);   //将画笔设为绿色
        for (int i=0; i<mSnake.getLength(); i++)
        {
            Body body = mSnake.getBody(i);
            g.fillRect(     body.col*TileWidth+XOffset,
                            body.row*TileHeight+YOffset,
                            TileWidth, TileHeight);
        }
    }
```

getBody 用来获取蛇的某一节，以一个循环遍历一条蛇的每一节，并绘制出来（fillRect 的作用是绘制实心矩形），就完成了整条蛇的绘制。这些坐标的确定应该是不难的。

最后，在 paint 函数里加入 drawSnake(g) 以完成对蛇绘制方法的调用。

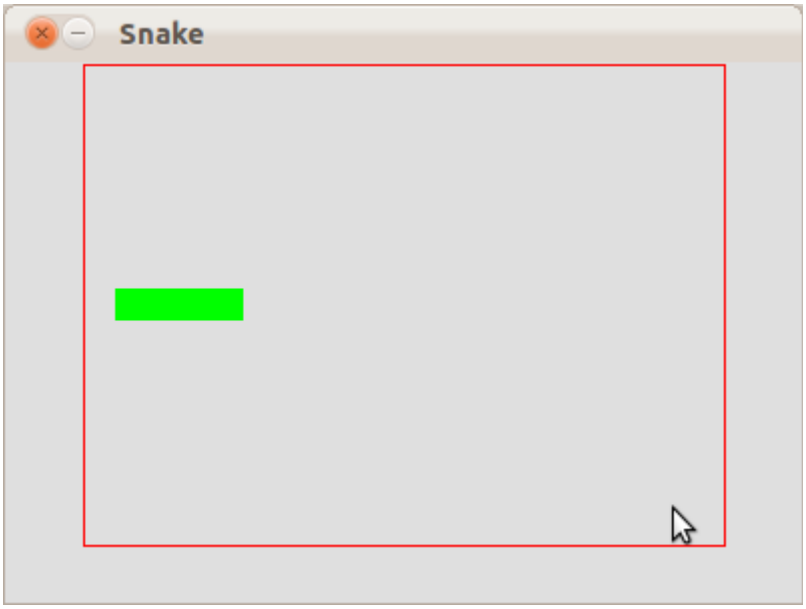
请仔细核对你的代码，使之通过编译并达到预期效果，这将是一个雏形，以后将不再出现整段的代码，所以务必核对仔细。

```
import java.awt.*;
import java.awt.event.*;
```

```
class Board extends Frame
{
	public Board(String title)
	{
		super(title);
		setSize(Width, Height);
		setVisible(true);
		setResizable(false);
		mSnake = new Snake();
	}
	public void drawDecoration(Graphics g)
	{
		g.setColor(Color.RED);
		g.drawRect(XOffset, YOffset, Column*TileWidth, Row*TileHeight);
	}
	public void paint(Graphics g)
	{
		drawSnake(g);
		drawDecoration(g);
	}

	public static void main(String args[])
	{
		Board board = new Board("Snake");
	}
	public void drawSnake(Graphics g)
	{
		g.setColor(Color.GREEN);
		for (int i=0; i<mSnake.getLength(); i++)
		{
			Body body = mSnake.getBody(i);
			g.fillRect(body.col*TileWidth+XOffset, body.row*TileHeight+YOffset,
TileWidth, TileHeight);
		}
	}
	private static final int Width = 400;
	private static final int Height = 300;
	private static final int TileWidth = 16;
	private static final int TileHeight = 16;
	public  static final int Row = 15;
	public  static final int Column = 20;
	private static final int XOffset = (Width - Column*TileWidth)/2;
	private static final int YOffset = (Height- Row*TileHeight)/2;

	private Snake mSnake;
}
```



这是一条蛇。。。。好吧，这货可能不是蛇。。。。是的，它是一条蠕虫。。。。

还是回来 Snake.java 中来，来完成一个最大的函数，MOVE～当然，move 不会是这一次写完的，先写一部分，它还要经过几次修订。

先来确定它的脑袋移动后会出现在什么地方，这个取决与两个因素，当前脑袋的位置，与及它的运动方向
定义一个变量 Body head，（注意没有实例化）还有一个 Body headold = mBody.get(0) 来表示当前脑袋对象。
然后要分情况讨论了，方向不同，head 坐标不同。

```
switch (mDirection)
{
case UP:
        head = new Body(headold.row-1, headold.col);
        break;
case DOWN:
        head = new Body(headold.row+1, headold.col);
        break;
case LEFT:
        head = new Body(headold.row, headold.col-1);
        break;
case RIGHT:
        head = new Body(headold.row, headold.col+1);
        break;
default:
        head = new Body(0, 0);
```

```
                break;
        }
```

为什么这样写是显而易见的，需要提一下的是最后那个 default，这个是永远不会被执行的，因为 mDirection 一共就四种可能，而且都被考虑到了，之所以加一个 default 是因为编译器任为可能有没考虑到的地方，这样 head 就没有被正常初始化了。

接下来把这个新脑袋加入这条蛇中

```
        mBody.add(0, head);
```

第一个参数是插入的位置，我们要把 head 插在头的位置，所以填 0，第二个参数为要添加的结点。

最后把尾巴移除即可，注意序列是从 0 开始的，所以要减 1。

```
        mBody.remove(mBody.size()-1);
```

OK，这样 Snake.java 再次靠一段落。现在再来看看怎么样能让这条蛇动起来吧。

比如我们想让这条蛇每半秒钟运动一次，该如何实现呢？当然可以利用计实器，但是为了涉及更广的方面，这里采用多线程的方法，让一个线程调用 move 方法后睡眠半秒钟，再接着调用 move，再睡半秒钟……

再来创建一个文件 Updater，并重写其中的 run 方法。

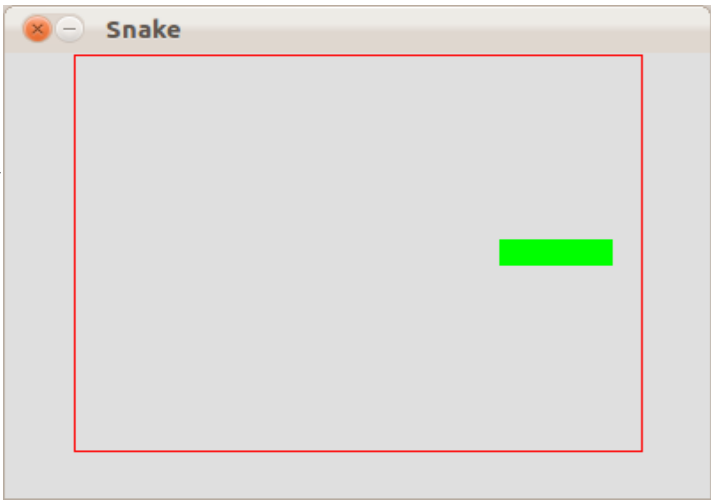
```
class Updater extends Thread
{
        public Updater(Board b, Snake s)
        {
                mBoard = b;
                mSnake = s;
        }
        public void run()
        {
                while (true)
                {
                        mSnake.move(mBoard);
                        mBoard.repaint();
                        try
                        {
                                sleep(500); //线程暂停 500 毫秒，详见 API 手册，这个函数会产生异常，这
```

里直接打印出来了，并未做进一步处理。

```
                } catch (Exception e)
                {
                        System.out.println(e.toString());
                }
        }
    }
    private Board mBoard;
    private Snake mSnake;
}
```

保存 Updater.java，打开 Board.java，添加一个 Updater 对象，这个 Updater 是负责刷新蛇以及刷新窗口的，所以属于 Board 类是逻辑上可行的。接着在构造函数里对它实例化，并运行这个线程

```
mUpdater = new Updater(mBoard, mSnake);
mUpdater.start();
```



编译运行一下，你的蛇已经能动了，但是画面有些闪烁，这是由于 repaint 是先擦除背景，再绘制新图的原因，这中间有一个空白的过程，解决这个问题的方法是采用双缓冲技术，不过这个不在本文讨论之内。

接下来来解决一个跨度有些大的问题，之所以放在这里解决，是因为它根真正接下来要解决的问题是一类问题，呵呵，不绕了，之前说过这个窗口无法被关闭，这是因为我们的 Board 无法对关闭窗口这一消息进行响应。就是说窗口虽然能知道有人点了关闭按钮，但点了不知道如何支做，比如酷狗音乐，你点击关闭会使音乐后台播放，而一般的窗口刚是退出。所以我们必须要对关闭这一事件进行处理。

在 Boar.java 文件的末尾，Board 类之外添加一个类

```
class BoardHandler extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}
```

这个 BoardHandler 从 WindowAdapter 类继承而来，而只需要对 windowClosing 方法进行重写，就可以达到对关闭消息进行处理的目的，因为它默认是什么也不做的。这里就一句话 System.exit(0);退出 java 虚拟机，不过这里原谅我，由于本人对 java 也不太熟，直接这样写会不会造成资源泄漏就不知道了，但它是退出程序最简单的方法。然后在 Board 类的构造函数里加上

```
addWindowListener(new BoardHandler());
```

这样 Board 就可以对关闭窗口的事件进行响应了。

（PS：这个 addWinowListener 方法实际是 Window 类的，Frame 从 Window 派生而来，说明 Frame 是一个 Window，Board 从 Frame 派生而来，说明 Board 是一个 Frame，也就是说 Board 是一个 Window，所以它也继承了 Window 的 addWindowListener 方法。）

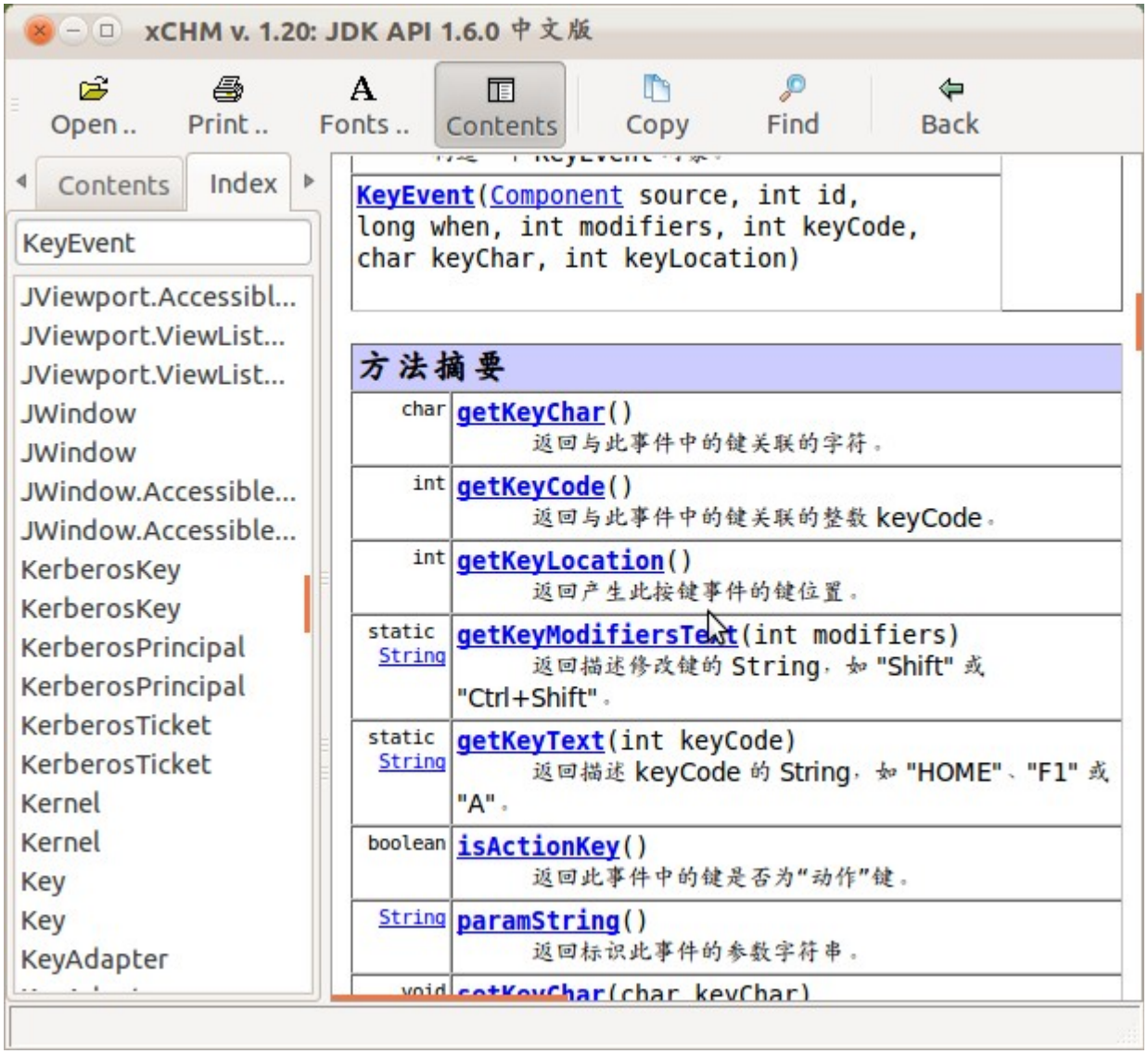
接下来，作为一个游戏，要对里面的东西进行控制，我们希望通过键盘来控制我们的蛇，同样的，刚才我们用 WindowListener 来监听窗口事件，我们一样可以用 KeyListener 来监听键盘事件。

在刚才类在下面再写一个类

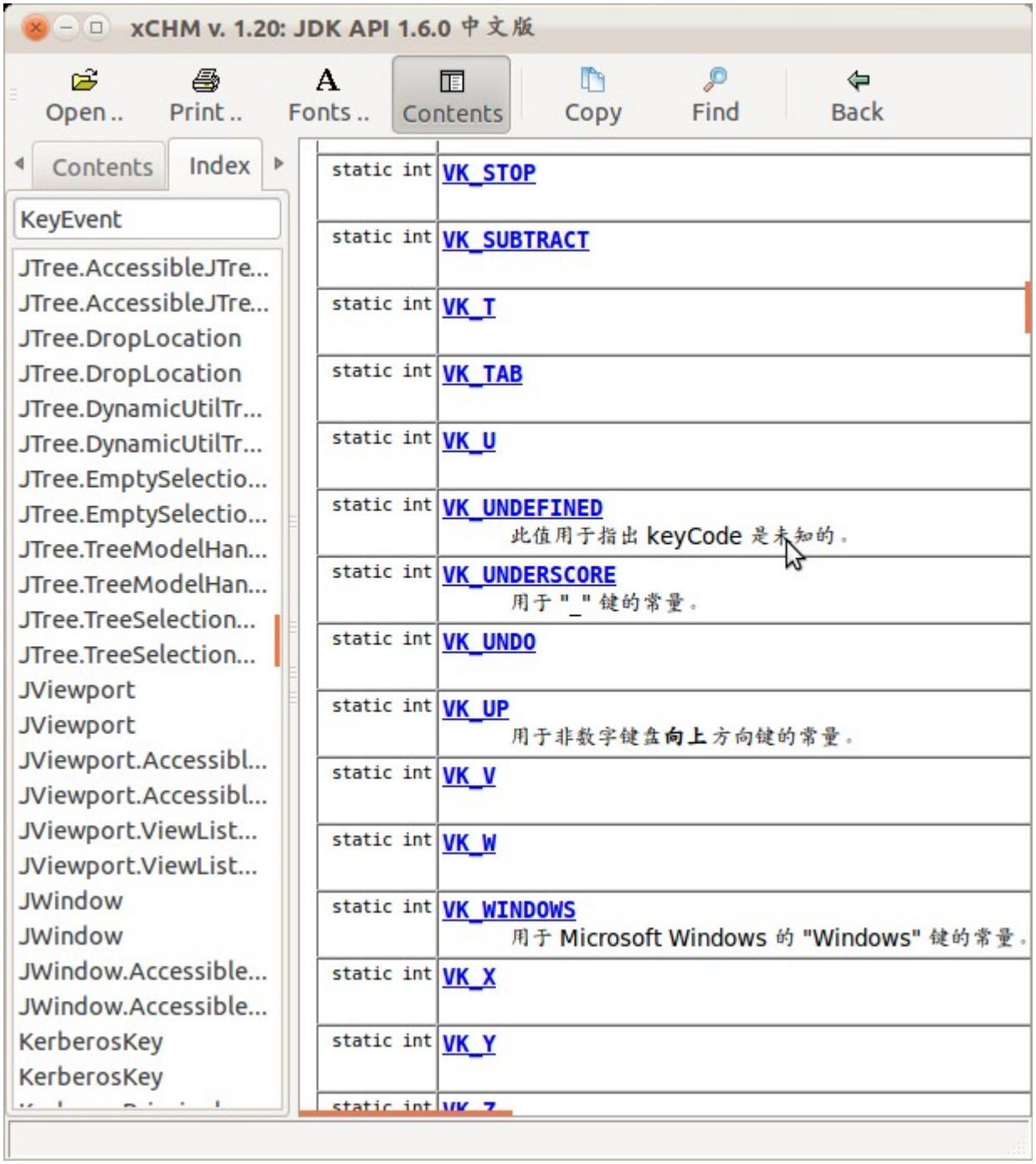
```
class KeyHandler extends KeyAdapter
{
	public KeyHandler(Snake s)
	{
		mSnake = s;
	}
	public void keyPressed(KeyEvent e)
	{
	}
	private Snake mSnake;
}
```

KeyHandler 是用来处理按键的，按键是操纵蛇的，所以必须拥有一个 Snake 对象。

重写其中的 keyPressed 函数就能对按键进行响应，这个函数传给了我们一个 KeyEvent 对像，查 API 可以看到如下页图所示内容。



注意其中的 getKeyCode，这是我们将要用到的，它返回一个整数 keyCode。KeyCode 是什么？在键盘上，有很多键，每一个键都与一个整数对应，也就是一个键到整数的映射。注意左 shift 和又 shift 是不同的。但又有问题了，如何知道一个键比如向上的键头与哪个整数对应？在 KeyEvent 类里定义了一些静态量，在刚才的页面上住上翻，可以看到如下内容



通过判断 e.getKeyCode() 是否等于 KeyEvent.VK_UP 就能判断被按下的是否是向上键。

这样，我们就可以完成 keyPressed 的函数。

```
public void keyPressed(KeyEvent e)
{
	switch (e.getKeyCode())
	{
	case KeyEvent.VK_UP:
		mSnake.setDirection(DIRECTION.UP);
		break;
	case KeyEvent.VK_DOWN:
		mSnake.setDirection(DIRECTION.DOWN);
		break;
	case KeyEvent.VK_LEFT:
		mSnake.setDirection(DIRECTION.LEFT);
		break;
	case KeyEvent.VK_RIGHT:
		mSnake.setDirection(DIRECTION.RIGHT);
		break;
	}
}
```

然后，同样在 Board 的构造方法中添加 addKeyListener(new KeyHandler(mSnake));

再次编译运行，现在你可以通过按键盘的上下左右来操纵这条蛇了。但是，等一下，这条蛇好像太过自由了，在向右走的时候按向左，蛇会“穿过”自己的身体，往回走，这是不被允许的。让我们回到 Snake.java 文件。找到 setDirection 函数，进行修改

```
	switch (d)
	{
	case UP:				//当向上走的时候，想把方向设为向下是不被允许的，下同
		if (mDirection == DIRECTION.DOWN)
		{
			return ;
		}
		break;
	case DOWN:
		if (mDirection == DIRECTION.UP)
		{
			return;
		}
		break;
	case LEFT:
		if (mDirection == DIRECTION.RIGHT)
		{
			return;
		}
		break;
	case RIGHT:
		if (mDirection == DIRECTION.LEFT)
		{
			return;
		}
```

```
                break;
                mDirection = d;
        }
```

但是这样真的行吗？再次运行就能发现问题，假设蛇在向右行进，这时按下向上，在蛇还没移到下一格的时候，再按下左，你同样会发现蛇“穿过”了自己的身体往回走。

如何解决呢？我的方法是，按键并不直接改变蛇的方向，而是先记下按键的方向，在 move 的时候再改变蛇的方向。在 Board 中再定义一个 private DIRECTION mNextDirection;来表示接下来走的方向。在 reset 函数中对它进行初始化

mNextDirection = DIRECTION.RIGHT;

然后将 setDirection 中的 mDirection = d; 改为 mNextDirection = d;

在 move 函数中的开头加入 mDirection = mNextDirection;

至此，蛇走起来正常了。这条蛇的控制系统也就全部完成了。

只有一条蛇是不是少了点什么？是的，下面来完成食物系统。先来为食物建一个类，新建文件 Food.java。但是这个 Food 类怎么设计，它表现了和 Body 极为类似，包含有一组坐标，经过考虑，我让这个 Food 类从 Body 类派生出来，说 Food 是一个 Body 也没错，当 Food 被吃下去的时候也就成了一个 Body，可以把它看成一个更自由的 Body。

来看看 Food 的定义

```
import java.util.Random;

class Food extends Body
{
        public Food()
        {
                super(0, 0);
                Random rnd = new Random();
                this.row = rnd.nextInt(Board.Row);
                this.col = rnd.nextInt(Board.Column);
        }
}
```

这里的构造函数不需要参数，直接用随机数来产生，当然前提是要在蛇的运动区域内。nextInt(n)将产生[0,n)的随机数。

Food 也是游戏盘的一部分，所以在 Board 加入

```
private Food mFood;
```

并添加一个函数

```
        public void drawFood(Graphics g)
        {
                g.setColor(Color.BLUE);
                g.fillRect(XOffset+mFood.col*TileWidth, YOffset+mFood.row*TileHeight, TileWidth,
TileHeight);
```

```
    }
```
用来画食物，当然别忘了在 paint 方法中支调用它。

再在 Board 中加一个函数 produceFood，用来产生一个新是食物对象。

```
    public Food produceFood()
    {
        boolean finished;
        Food food;
        do
        {
            finished = true;
            food = new Food();
            for (int i=0; i<mSnake.getLength(); i++)
            {
                if (mSnake.getBody(i).col == food.col && mSnake.getBody(i).row ==
```
food.row) //如果产生食物的位置已经有蛇的某节在上面了，则生成失败，将 finished 设为 false 并再次产生。
```
                {
                    finished = false;
                }
            }
        } while (!finished);
        mFood = food;
        return food;
    }
```

注意这个方法是完全产生了一个新对象，而不是将重设 mFood 的坐标。然后在 Board 的构造方法中来调用 produceFood 来创建一个食物，但它的位置是比较重要的，你需要放在 mSnake 产生之后，mUpdater 调用 start 之前。如果在 snake 产生之前调用的话，snake 可能会盖在食物上。如果在线程启动之后调用的话，线程里会调用 move，而 move 里又使用了 mFood 对象，这时的 mFood 是没有被初始化的。

现在，让我们来对 move 函数再进行一次修订，这次是修订是在末尾处，如果走到位置有一块食物，那就让食物做为新的脑袋，并且不移除尾巴，这样就使得蛇的长度 + 1。要在 Snake 类中判断脑袋是否与食物重合，那就要知道食物的坐标，但是你会发现缺少一个对食物访问的方法，所以在修订 move 之前，先在 Board 类中加入一个方法

```
    public Food getFood()
    {
        return mFood;
    }
```

回到 Snake 的 move 函数中。在得到了 head 后，来做一个判断，看是否吃到了食物

```
        Food f = b.getFood();
        if (f.row == head.row && f.col == head.col)
        {
            mBody.add(0, f);        //Food 是从 Body 中派生出的，可以隐式转换为 Body 类型
            b.produceFood();        //吃到食物后要立刻产生一个新的食物
```

```
		}
		else
		{
			mBody.remove(mBody.size()-1);
			mBody.add(0, head);
		}
```

完成了这些代码之后，你的蛇就可以靠吃变长了。

最后一步，判断游戏的终止条件，也就是蛇撞到了墙或自己的身体，这由两部分组成，这两个功能应该属于 Snake 类，让蛇自己判断自己是否还活着。在 Snake 类中添加两个函数

public boolean checkBodyCollision() 和 public boolean checkBoundCollision(Board b)

对墙的检查需要一个参数 Board，因为蛇并不知道外界的情况。

两个函数都是如果发现了碰撞就返回 true，而且我们是每走一步都要判断是否发生了碰撞，所以每次只要检察头部就好了。

```
	public boolean checkBodyCollision()
	{
		Body head = getBody(0);

		for (int i=1; i<getLength(); i++)
		{
			if (head.row == getBody(i).row && head.col == getBody(i).col)
			{
				return true;
			}
		}
		return false;
	}
	public boolean checkBoundCollision(Board b)
	{
		Body head = getBody(0);
		if (head.col >= b.Column || head.col < 0 || head.row < 0 || head.row >= b.Row)
		{
			return true;
		}
		return false;
	}
```

现在再来对 move 进行一次修订，在 move 的末尾添加

```
if (checkbodycollision() || checkBodyCollision(b))
{
}
```

等等……如果蛇挂了，会怎么样呢？这条 if 里写些什么？这取决于你如何处理挂了这一事件。这里，简单的提示一段挂了的信息，并让玩家按 F2 重新开始。

如何让游戏盘正确的显示挂了的信息或是正常的游戏画面呢？给 Board 加一个状态量，之前提过，状态量一般用枚举类型，所以新建一个文件 STATE.java，因为这次要在多个类中使用这个枚举，所以要单独写在一个文件中让其它类访问。

STATE.JAVA 十分的简单，这里添加两个状态，正常运行，和死亡。如果你不满意这几个状态，以后可以添加 PAUSE 之类的，实现起来也是很方便的。

```
enum STATE
{
	RUN, DEAD;
}
```

按之前说的，在 Board 中添加一个状态量，private STATE mState;

然后添加一个 set 方法，

```
	public void setState(STATE s)
	{
		mState = s;
	}
```

并在构造函数调用 setState(STATE.RUN);以使游戏开始时是运行的。

接下来再写一个显示挂了的提示信息的函数

```
	public void drawDeadMessage(Graphics g)
	{
		g.drawString("YOU DEAD, Press F2 to retry~~~~", XOffset, Height/2);
	}
```

然后把它放在 paint 函数里让它显示出来，但是只有满足条件（蛇已经挂了的时候）才能让它显示，所以在 paint 函数中加一个判断

```
	public void paint(Graphics g)
	{
		switch (mState)
		{
		case RUN:
			drawSnake(g);
			drawFood(g);
			drawDecoration(g);
			break;
		case DEAD:
			drawDeadMessage(g);
			break;
		}
	}
```

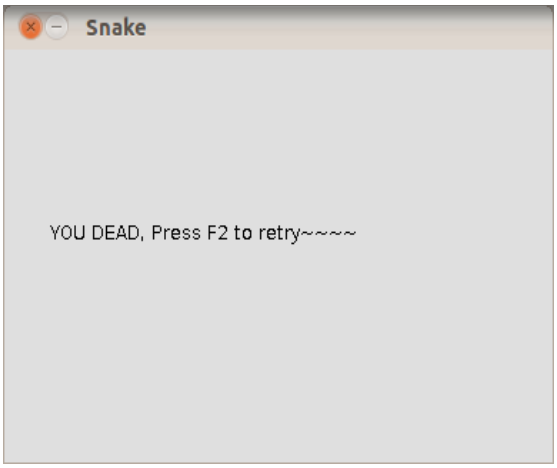
下面把 move 中那个 if 语句补全，也是最后一次修订。

```
		if (checkBodyCollision() || checkBoundCollision(b))
		{
			b.setState(STATE.DEAD);
		}
```

现在蛇挂了，就会提示一段信息，但是无法重新开始了

下面就来解决这最后一个问题。

找到 Board.java KeyHandler 这个类，我们需要对按下 F2 进行响应——将游戏设为正常状态并重置蛇。很幸运，这两个功能都有相应的方法来实现，但是 KeyHandler 要稍做改动，给它加一个成员变量

```
private Board mBoard;
```
并修改它的构造函数
```
        public KeyHandler(Board b, Snake s)
        {
                mBoard = b;
                mSnake = s;
        }
```
在 keyPressed 事件处理这新加一个 case
```
        case KeyEvent.VK_F2:
                mSnake.reset();
                mBoard.setState(STATE.RUN);
                break;
```


最后也别忘了在 Board 的构造函数中修改创造 KeyHandler 的参数
```
addKeyListener(new KeyHandler(this, mSnake));
```

至此，一个完整的贪食蛇游戏就完成了。

后记

我也是这个学期才接触 java 的，参考的东西也就只是教材，实验指导，以及 API 手册。如果有什么错误，请以代码为主。书写代码时请务必注意缩进，格式非常的重要，那一函数之前那些空的并不是一堆空格，而是几个制表符（Tab 键），不会的童鞋要注意下了。另外，本文所有代码都是用文本编辑器写的，用 javac 编译的，如果你不会用 Eclipse 或 JCreator 之类的 IDE 也别找我，因为我也不会。那些成员变量的命名你会发现都是 mXXXX，这里的 m 代表 member，个人有个人的命名习惯，如果你觉得这种命名你还能接受，就请采纳之。

由及大似然思想，随便一个完整的程序用到的知识也将是以后最常用到的，它所包含的内容与考试应该大致是对应的，要复习的童鞋可以看看。

时间仓促，错误难免，有任何问题可以发邮件到 Pz_L@yahoo.cn