Synchronization & Game Loop Design
class TutorialThread extends Thread {

    @Override public void run() {
        Canvas c;
        while (_run) {
            c = null;
            try {
                c = _surfaceHolder.lockCanvas(null);
                synchronized (_surfaceHolder) {
                    _panel.updatePhysics();
                    _panel.onDraw(c);
                }
            } finally {
                // do this in a finally so that if an exception is thrown
                // during the above, we don't leave the Surface in an
                // inconsistent state
                if (c != null) {
                    _surfaceHolder.unlockCanvasAndPost(c);
                }
            }
        }
    }
}
Code Examination – SurfaceView onTouch ()

@Override
public boolean onTouchEvent(MotionEvent event) {
    synchronized (_thread.getSurfaceHolder()) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            GraphicObject graphic = new GraphicObject(
                BitmapFactory.decodeResource(getResources(),
                R.drawable.icon));
            graphic.getCoordinates().setX((int) event.getX() -
                graphic.getGraphic().getWidth() / 2);
            graphic.getCoordinates().setY((int) event.getY() -
                graphic.getGraphic().getHeight() / 2);
            _graphics.add(graphic);
        }
        return true;
    }
}

From:
Questions to think about

1. What is the purpose of 
c = _surfaceHolder.lockCanvas(null);
2. What is the purpose of synchronized?
3. Where do we have to use synchronized?
4. What threads exist and what are they doing?
Back to SurfaceView

• Provides a dedicated surface for a secondary thread to render screen content
• All SurfaceView and SurfaceHolder.Callback methods are called from the thread running the SurfaceViews window (typically the main application thread)

What potential thread problems can exist?
Synchronization

• Every Java object (including every class loaded) has an associated lock

• synchronized block
  – compiler adds instructions to acquire lock before executing code
  – compiler adds instructions to release lock after executing code

• thread owns the lock
More Synchronization

If thread A and thread B both have access to a Counter object and thread A owns the lock, thread B must wait for thread A to release the lock. Thus, simultaneous calls to increment and decrement behave correctly.

```java
public class Counter {
    private int count = 0;
    public void increment () {
        synchronized (this) {
            ++count;
        }
    }
    public void decrement () {
        synchronized (this) {
            --count;
        }
    }
}
```
Questions to think about

1. What is the purpose of \( c = \_\text{surfaceHolder}.\text{lockCanvas}(\text{null}); \)?
2. What is the purpose of \textit{synchronized}?
3. Where do we have to use \textit{synchronized}?
4. What threads exist and what are they doing?
Game Loop Design

• Games consist of:
  – getting user input
  – updating the game state (physics)
  – game AI
  – music/sound effects
  – game display
while (bIsRunning)
{
    updateGame ();
    drawGame ();
}
Terminology

• Frames Per Second (FPS) – number of times drawGame () is called

• Game Speed (GS) – number of times updateGame () is called