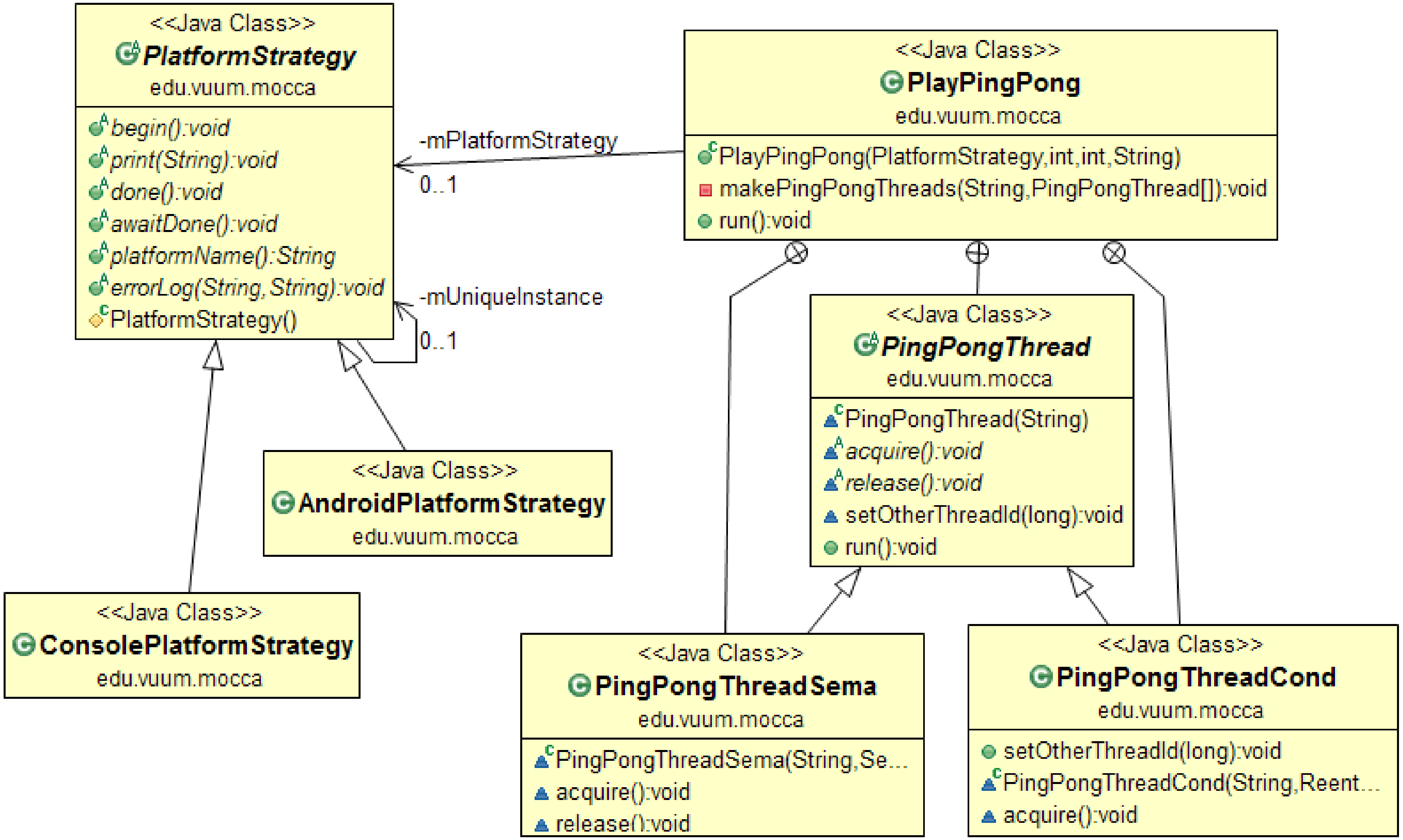
Android Concurrency: Java Synchronization & Scheduling Example (Part 2)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

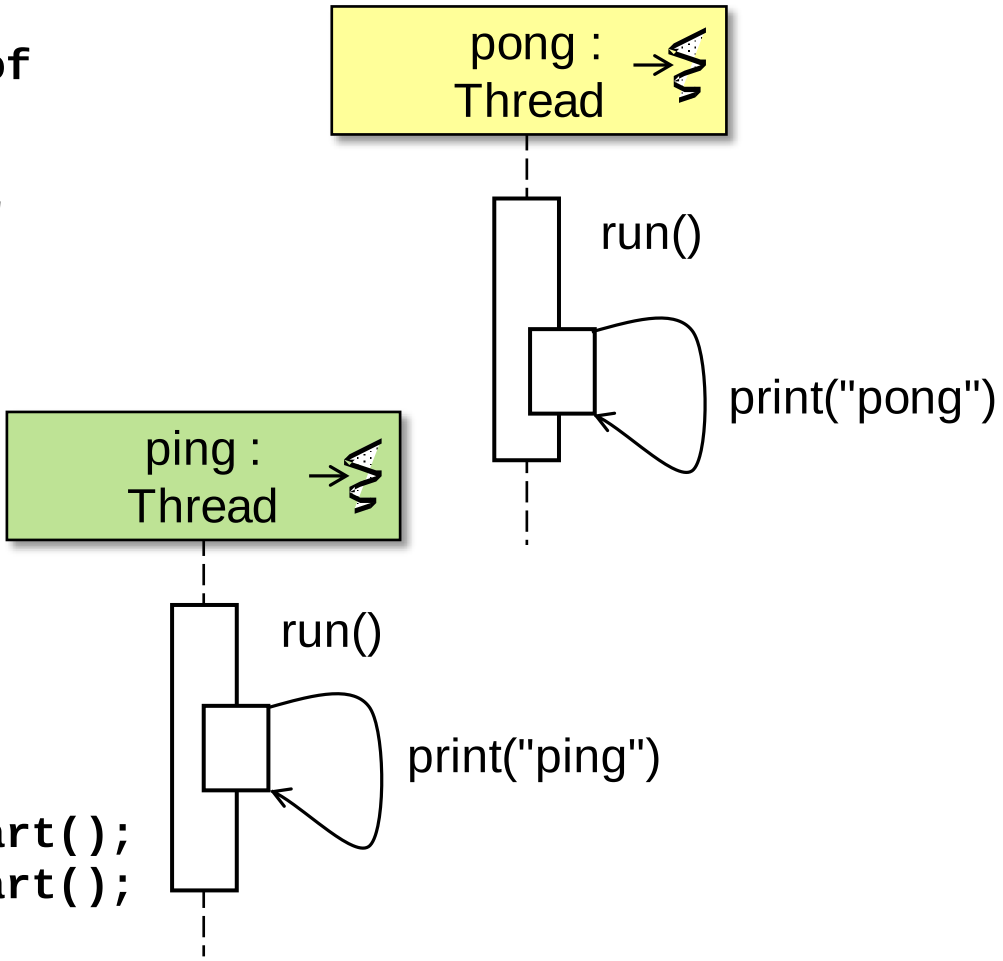
- Understand the implementation of our pattern-oriented & framework-based ping-pong program



Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code

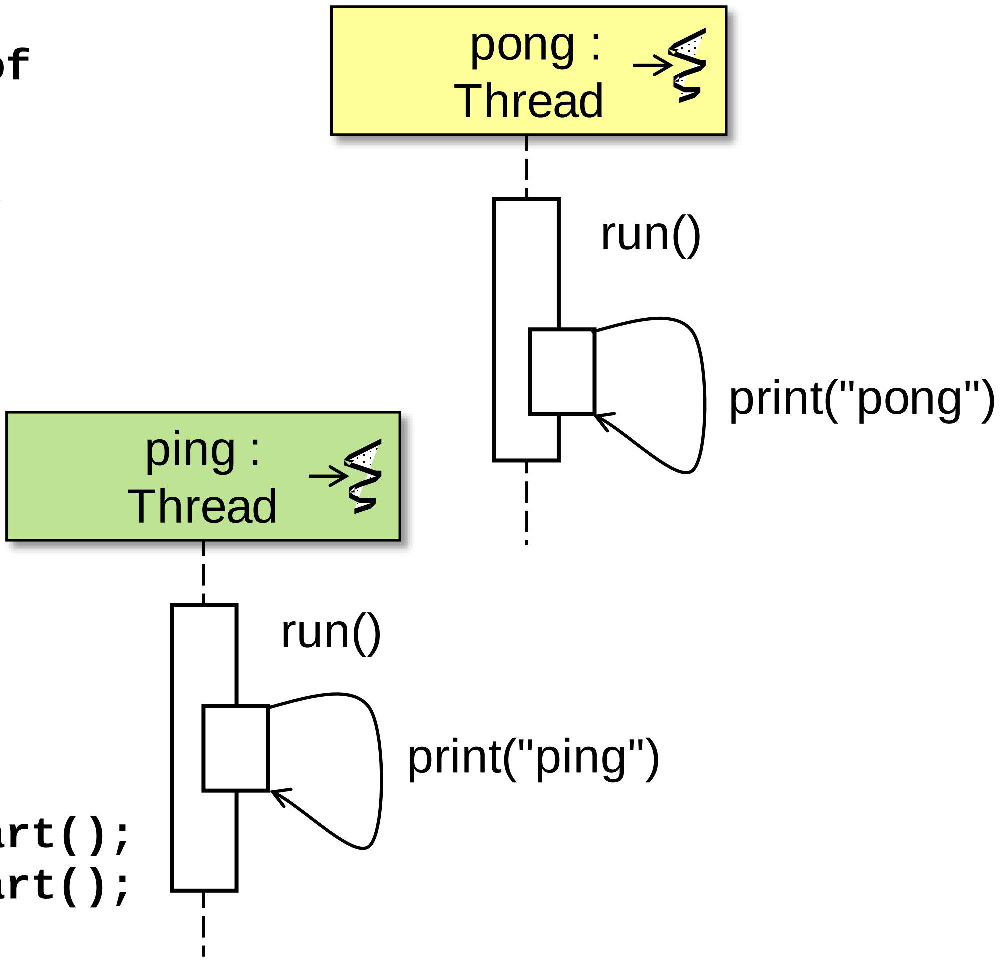
```
/**
 * Create the appropriate type of
 * threads with the designated
 * scheduling mechanism (e.g.,
 * "SEMA" for Semaphores, "COND"
 * for ConditionObjects, etc.).
 */
makePingPongThreads
  (mSyncMechanism,
   pingPongThreads);

/**
 * Start ping and pong threads,
 * which calls their run()
 * hook methods.
 */
pingPongThreads[PING_THREAD].start();
pingPongThreads[PONG_THREAD].start();
```



Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code

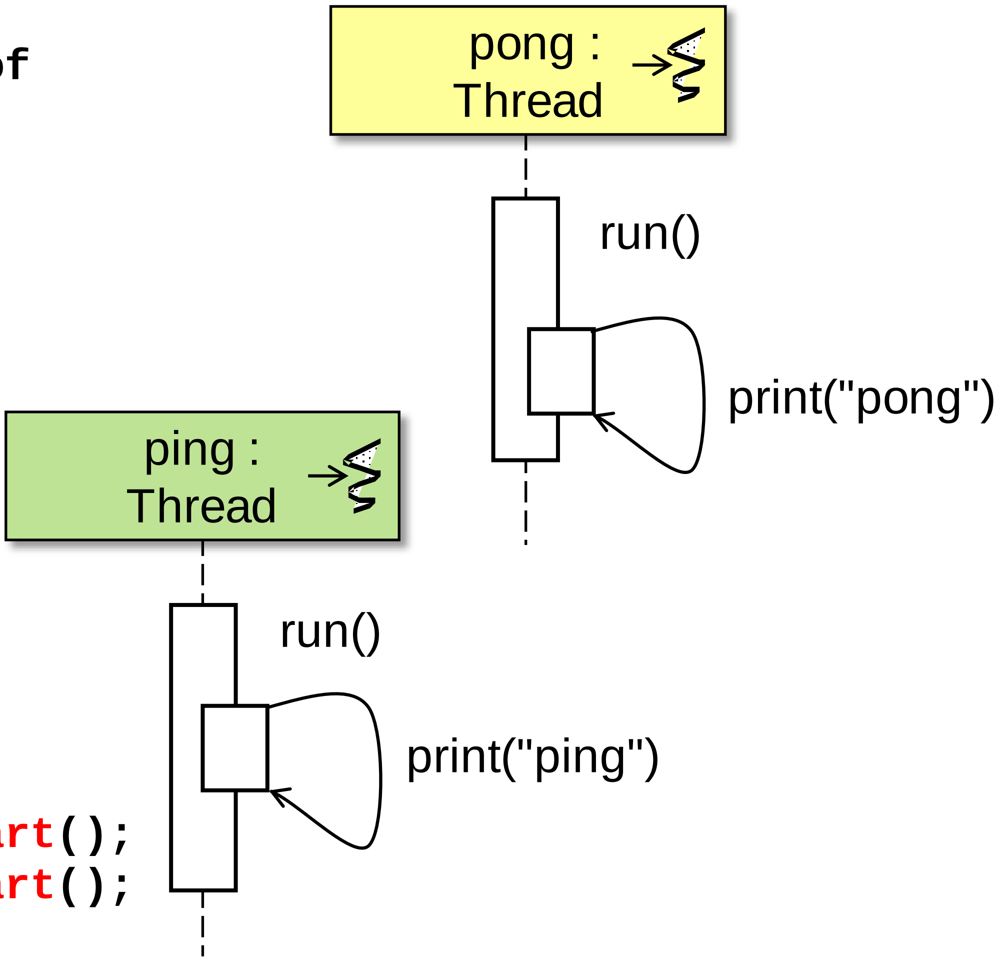
```
/**
 * Create the appropriate type of
 * threads with the designated
 * scheduling mechanism (e.g.,
 * "SEMA" for Semaphores, "COND"
 * for ConditionObjects, etc.).
 */
makePingPongThreads
  (mSyncMechanism,
   pingPongThreads);

/**
 * Start ping and pong threads,
 * which calls their run()
 * hook methods.
 */
pingPongThreads[PING_THREAD].start();
pingPongThreads[PONG_THREAD].start();
```



Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code

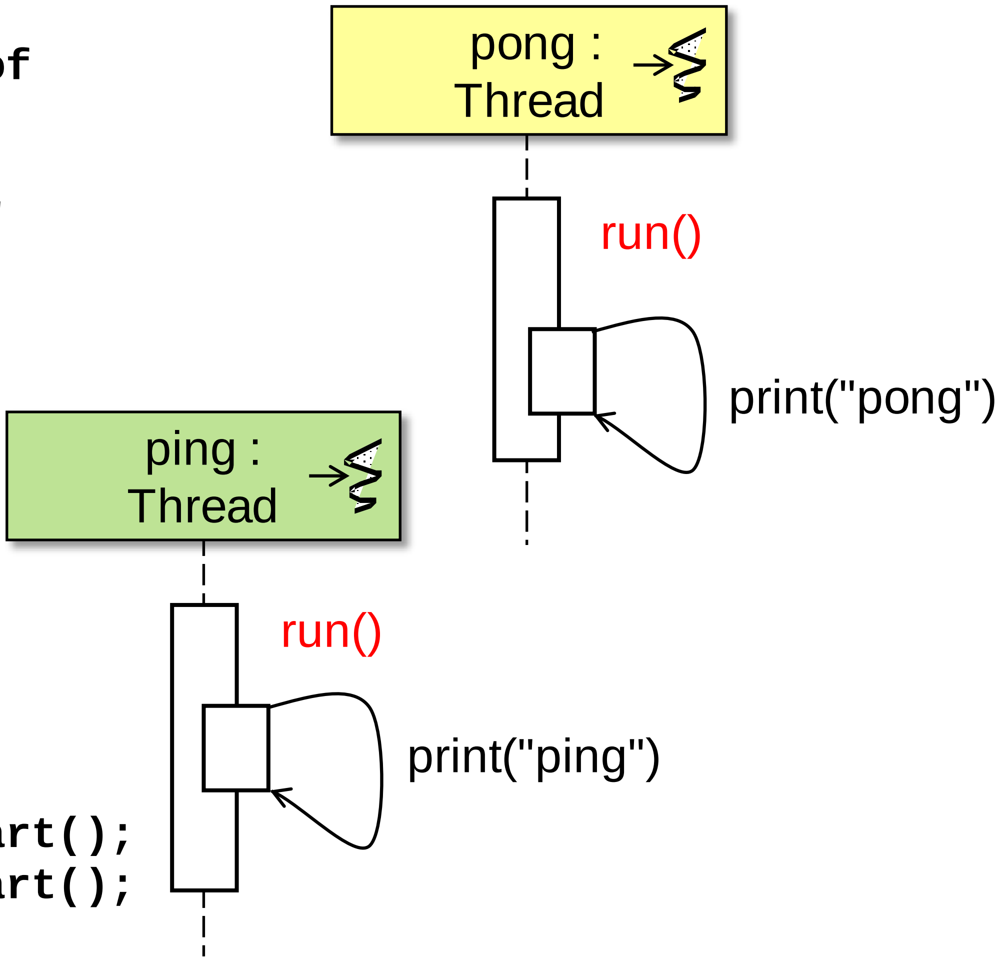
```
/**
 * Create the appropriate type of
 * threads with the designated
 * scheduling mechanism (e.g.,
 * "SEMA" for Semaphores, "COND"
 * for ConditionObjects, etc.).
 */
makePingPongThreads
  (mSyncMechanism,
   pingPongThreads);

/**
 * Start ping and pong threads,
 * which calls their run()
 * hook methods.
 */
pingPongThreads[PING_THREAD].start();
pingPongThreads[PONG_THREAD].start();
```



Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code

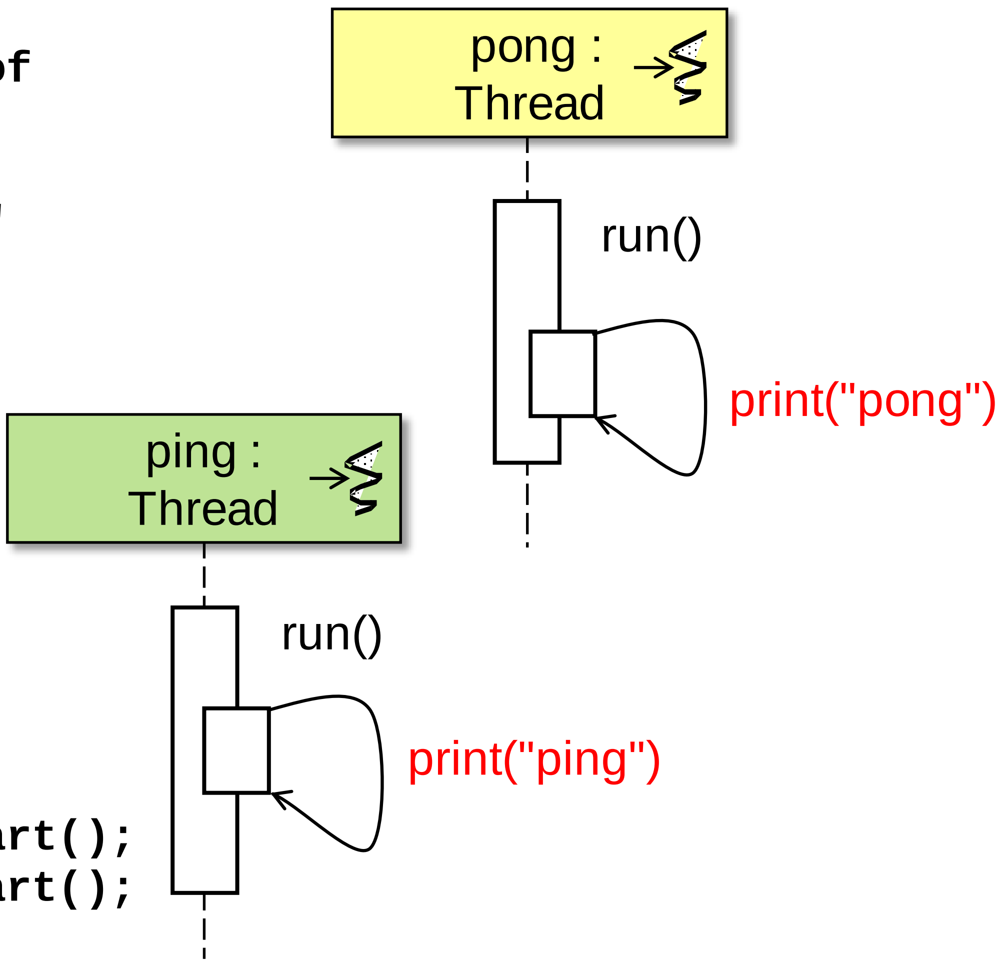
```
/**
 * Create the appropriate type of
 * threads with the designated
 * scheduling mechanism (e.g.,
 * "SEMA" for Semaphores, "COND"
 * for ConditionObjects, etc.).
 */
makePingPongThreads
  (mSyncMechanism,
   pingPongThreads);

/**
 * Start ping and pong threads,
 * which calls their run()
 * hook methods.
 */
pingPongThreads[PING_THREAD].start();
pingPongThreads[PONG_THREAD].start();
```



Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code

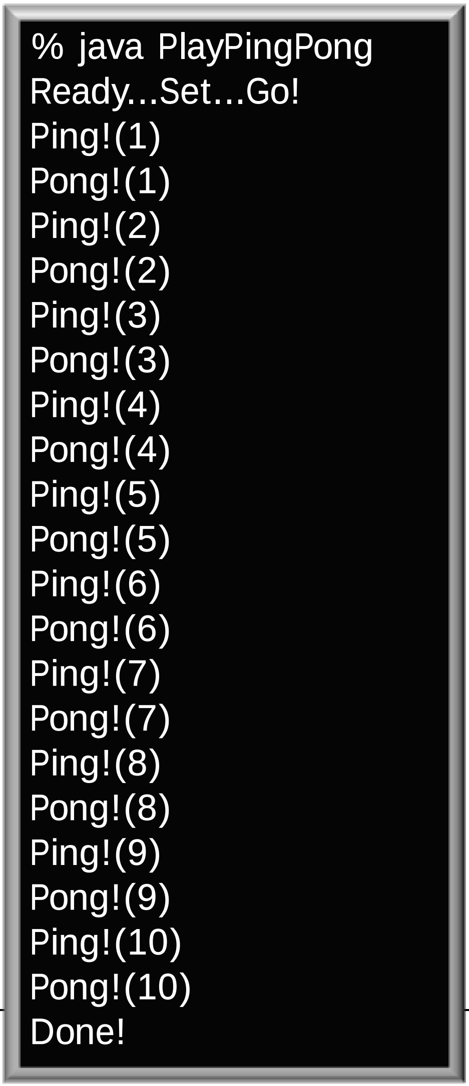
```
/**
 * Create the appropriate type of
 * threads with the designated
 * scheduling mechanism (e.g.,
 * "SEMA" for Semaphores, "COND"
 * for ConditionObjects, etc.).
 */
makePingPongThreads
  (mSyncMechanism,
   pingPongThreads);

/**
 * Start ping and pong threads,
 * which calls their run()
 * hook methods.
 */
pingPongThreads[PING_THREAD].start();
pingPongThreads[PONG_THREAD].start();
```



Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code

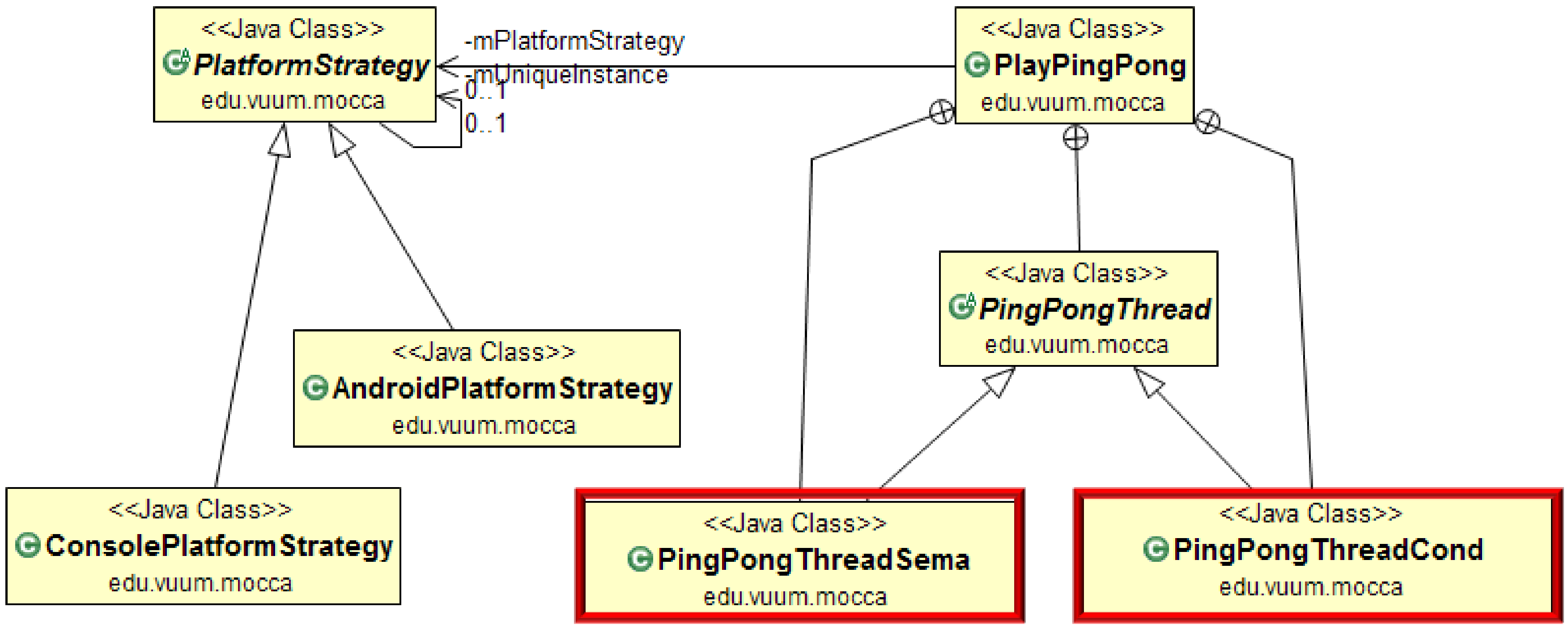
```
/**
 * Create the appropriate type of
 * threads with the designated
 * scheduling mechanism (e.g.,
 * "SEMA" for Semaphores, "COND"
 * for ConditionObjects, etc.).
 */
makePingPongThreads
  (mSyncMechanism,
   pingPongThreads);

/**
 * Start ping and pong threads,
 * which calls their run()
 * hook methods.
 */
pingPongThreads[PING_THREAD].start();
pingPongThreads[PONG_THREAD].start();
```



```
% java PlayPingPong
Ready...Set...Go!
Ping!(1)
Pong!(1)
Ping!(2)
Pong!(2)
Ping!(3)
Pong!(3)
Ping!(4)
Pong!(4)
Ping!(5)
Pong!(5)
Ping!(6)
Pong!(6)
Ping!(7)
Pong!(7)
Ping!(8)
Pong!(8)
Ping!(9)
Pong!(9)
Ping!(10)
Pong!(10)
Done!
```

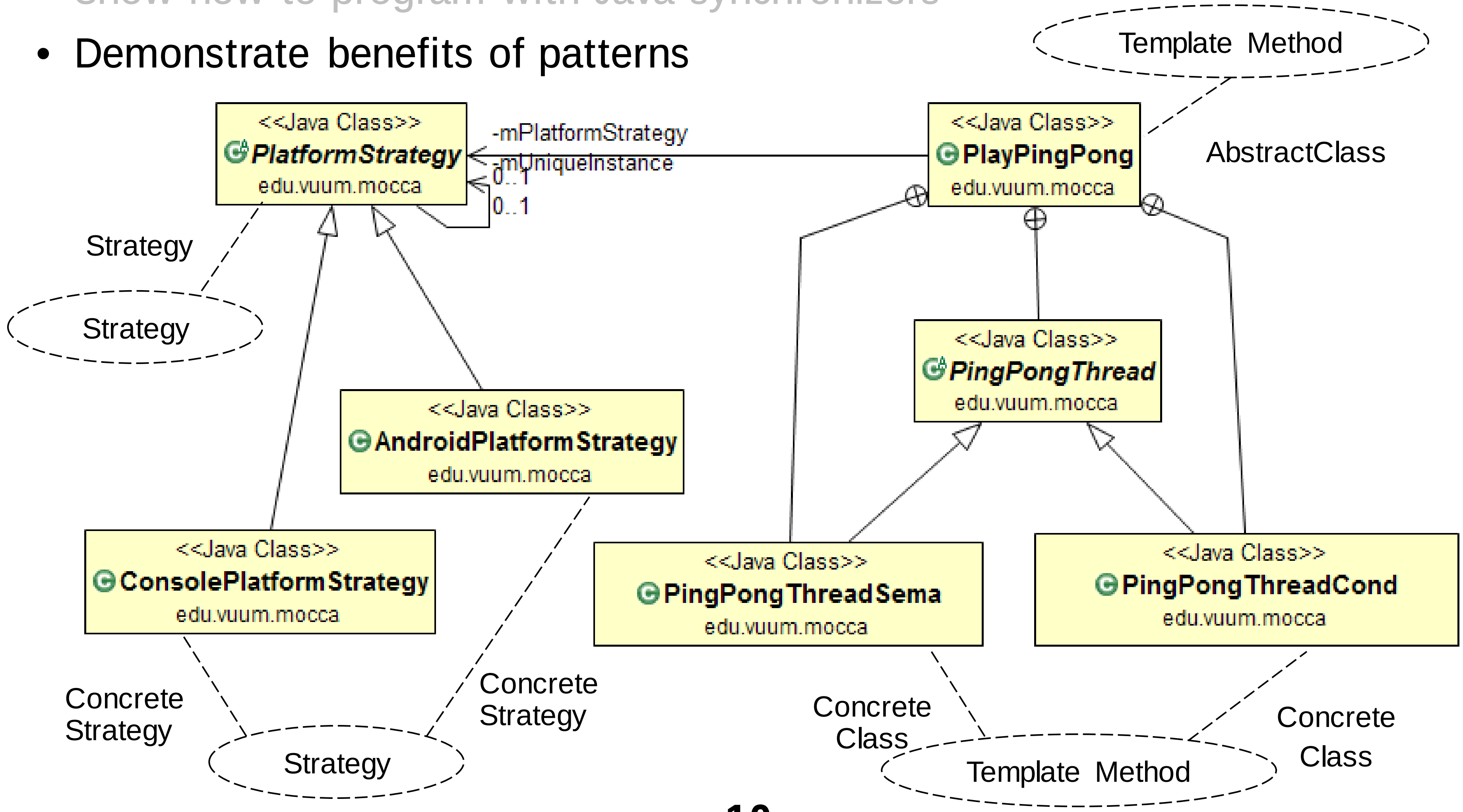
Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code
  - Show how to program with Java synchronizers



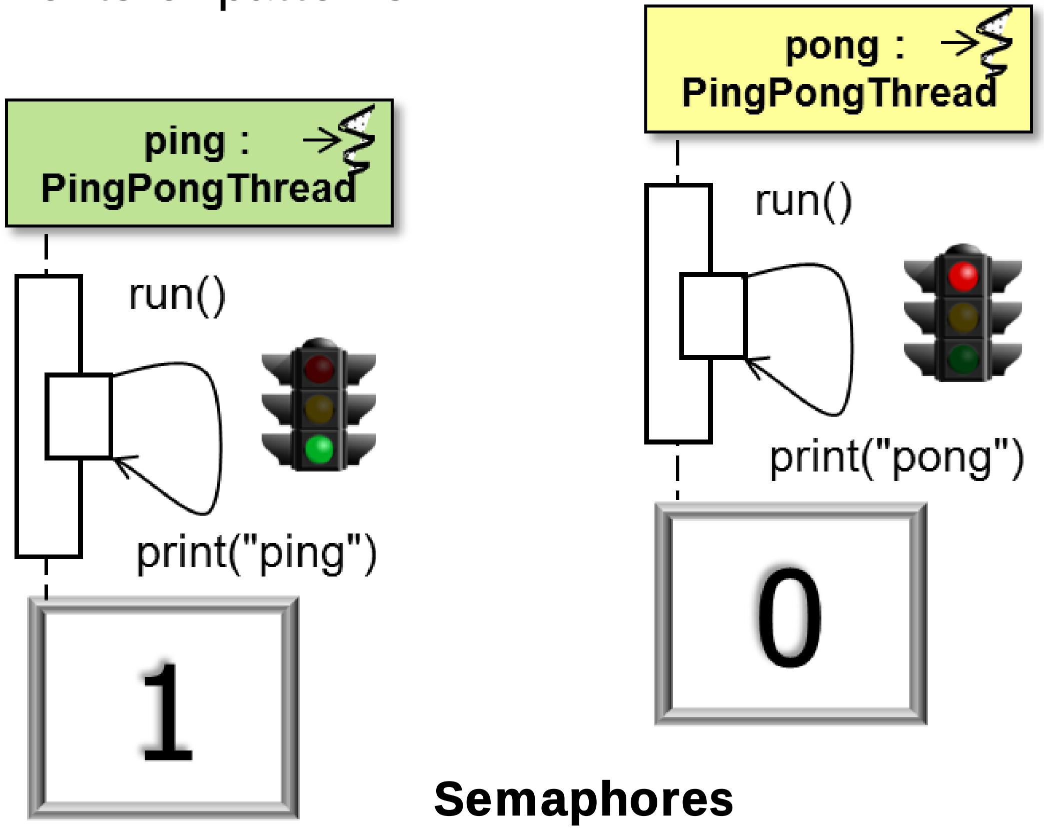
Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code
  - Show how to program with Java synchronizers
  - Demonstrate benefits of patterns



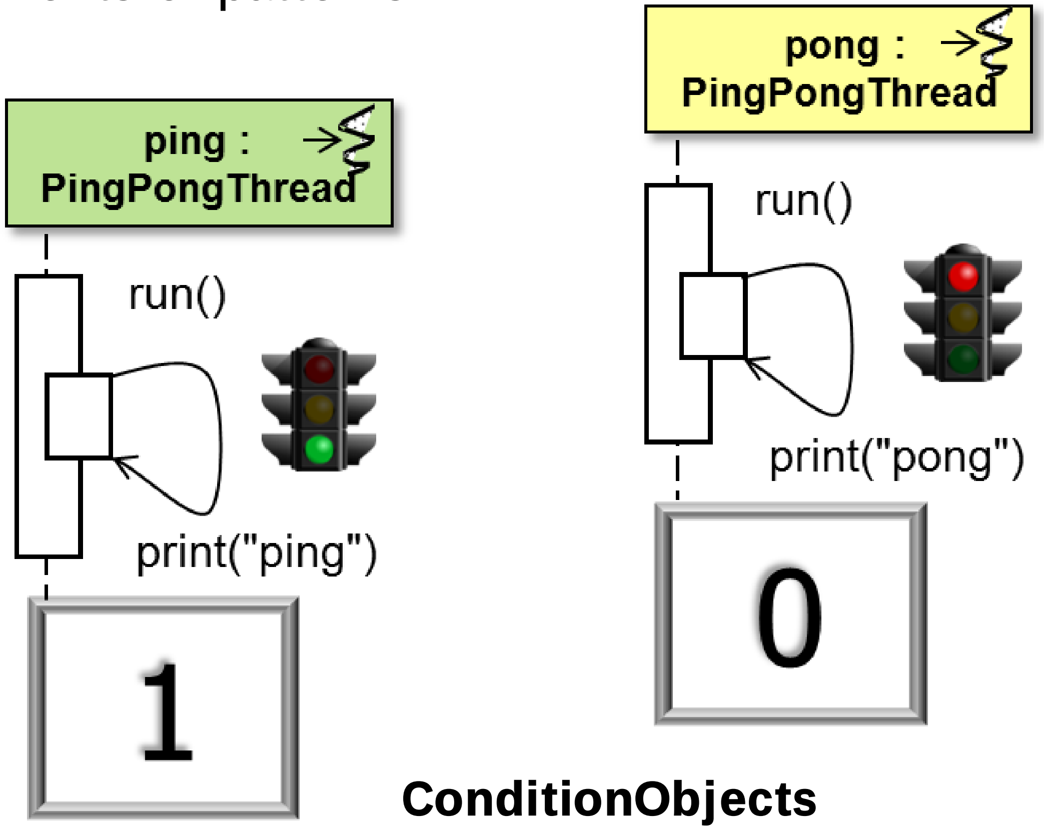
Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code
  - Show how to program with Java synchronizers
  - Demonstrate benefits of patterns



Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code
  - Show how to program with Java synchronizers
  - Demonstrate benefits of patterns



These changes don't affect the ping-pong algorithm or the framework software

Analyzing Pattern-Oriented Framework Solution

- We walk through the pattern-oriented framework source code
- It's important to run/read the code & (re)watch the video carefully to understand how it works

See github.com/douglascraigschmidt/POSA-14/tree/master/ex/PingPong/console

Implementation of the PlayPingPong Class

Implementation of the Ping-Pong Program

- Implementation of the PlayPingPong class

```
public class PlayPingPong implements Runnable {
```

Implementation of the Ping-Pong Program

- Implementation of the PlayPingPong class

```
public class PlayPingPong implements Runnable {
  ...
```

Provides a pattern-oriented framework for varying certain aspects of the ping/pong algorithm concurrently

Implementation of the Ping-Pong Program

- Implementation of the PlayPingPong class

```
public class PlayPingPong implements Runnable {
  private static volatile int
    mMaxIterations;

  private static volatile int
    mMaxTurns;

  private static volatile
    PlatformStrategy mPlatformStrategy;

  private static String mSyncMEchanism = "SEMA";
  ...
```

Implementation of the Ping-Pong Program

- Implementation of the PlayPingPong class

```
public class PlayPingPong implements Runnable {
  private static volatile int
    mMaxIterations;

  private static volatile int
    mMaxTurns;

  private static volatile
    PlatformStrategy mPlatformStrategy;

  private static String mSyncMEchanism = "SEMA";
  ...
```

Number of iterations to play ping/pong

Implementation of the Ping-Pong Program

- Implementation of the PlayPingPong class

```
public class PlayPingPong implements Runnable {
  private static volatile int
    mMaxIterations;

  private static volatile int
    mMaxTurns;

  private static volatile
    PlatformStrategy mPlatformStrategy;

  private static String mSyncMEchanism = "SEMA";
  ...
```

Number of times to print either "ping" & "pong" during each turn

Implementation of the Ping-Pong Program

- Implementation of the PlayPingPong class

```
public class PlayPingPong implements Runnable {
  private static volatile int
    mMaxIterations;

  private static volatile int
    mMaxTurns;

  private static volatile
    PlatformStrategy mPlatformStrategy;

  private static String mSyncMEchanism = "SEMA";
  ...
```

Strategy object used to print & synchronize completion of the "ping" & "pong" output

Implementation of the Ping-Pong Program

- Implementation of the PlayPingPong class

```
public class PlayPingPong implements Runnable {
  private static volatile int
    mMaxIterations;

  private static volatile int
    mMaxTurns;

  private static volatile
    PlatformStrategy mPlatformStrategy;

  private static String mSyncMEchanism = "SEMA";
  ...
```

Synchronization mechanism to use, e.g. Semaphore or ConditionObject

Implementation of the Ping-Pong Program

- Implementation of the PlayPingPong class

```
public class PlayPingPong implements Runnable {
  ...

  public PlayPingPong (PlatformStrategy platformStrategy,
                       int maxIterations, int maxTurns,
                       String syncMechanism) {
    mPlatformStrategy = platformStrategy;
    mMaxIterations = maxIterations;
    mMaxTurns = maxTurns;
    mSyncMechanism = syncMechanism;
  }
```

Constructor stores parameters used in the ping/pong algorithm

Implementation of the Ping-Pong Program

- Implementation of the PingPongThread class

```
public class PlayPingPong implements Runnable {
  ...
  static abstract class PingPongThread
                                extends Thread {

    abstract void acquire();
    abstract void release();




    private String nStringToPrint;



    PingPongThread(String stringToPrint)
    { mStringToPrint = stringToPrint; }
```

Implementation of the Ping-Pong Program

- Implementation of the PingPongThread class

```
public class PlayPingPong implements Runnable {
  ...
  static abstract class PingPongThread
                      extends Thread {

    abstract void acquire();
    abstract void release();




    private String nStringToPrint;



    PingPongThread(String stringToPrint)
    { mStringToPrint = stringToPrint; }
```

Plays role of "abstract class" in Template Method pattern

Implementation of the Ping-Pong Program

- Implementation of the PingPongThread class

```
public class PlayPingPong implements Runnable {
  ...
  static abstract class PingPongThread
                        extends Thread {

    abstract void acquire();
    abstract void release();
```

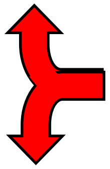
Abstract methods overridden by subclasses to schedule the order of printing ping & pong in run() template method

```
    private String nStringToPrint;



    PingPongThread(String stringToPrint)
    { mStringToPrint = stringToPrint; }
```

Implementation of the Ping-Pong Program

- Implementation of the PingPongThread class

```
public class PlayPingPong implements Runnable {
  ...
  static abstract class PingPongThread
                        extends Thread {

    abstract void acquire();
    abstract void release();





    private String nStringToPrint;



    PingPongThread(String stringToPrint)
    { mStringToPrint = stringToPrint; }
```


Define/set common data member

Implementation of the Ping-Pong Program

- Implementation of the PingPongThread class

```
public class PlayPingPong implements Runnable {
  ...
  static abstract class PingPongThread
                       extends Thread {



    public void run() {
      for (int loopsDone = 1;
           loopsDone <= mMaxIterations;
           ++loopsDone) {
        acquire();
        mPlatformStrategy.print(mStringToPrint
                               + "(" + loopsDone + ")");
        release();
      }
      mPlatformStrategy.done();
    }
```

Executes in a separate thread & plays role of the "template method" in the Template Method pattern

Implementation of the Ping-Pong Program

- Implementation of the PingPongThread class

```
public class PlayPingPong implements Runnable {
  ...
  static abstract class PingPongThread
                      extends Thread {


    public void run() {
      for (int loopsDone = 1;
           loopsDone <= mMaxIterations;
           ++loopsDone) {
        acquire();
        mPlatformStrategy.print(mStringToPrint
                         + "(" + loopsDone + ")");
        release();
      }
      mPlatformStrategy.done();
    }
```

Executes in a separate thread & plays role of the "template method" in the Template Method pattern

Implementation of the Ping-Pong Program

- Implementation of the PingPongThread class

```
public class PlayPingPong implements Runnable {
  ...
  static abstract class PingPongThread
                        extends Thread {


    public void run() {
      for (int loopsDone = 1;
           loopsDone <= mMaxIterations;
           ++loopsDone) {
        acquire();
        mPlatformStrategy.print(mStringToPrint
                               + "(" + loopsDone + ")");
        release();
      }
      mPlatformStrategy.done();
    }
```

Perform protocol that alternates printing "ping" or "pong"

Implementation of the Ping-Pong Program

- Implementation of the PingPongThread class

```
public class PlayPingPong implements Runnable {
  ...
  static abstract class PingPongThread
                        extends Thread {


    public void run() {
      for (int loopsDone = 1;
           loopsDone <= mMaxIterations;
           ++loopsDone) {
        acquire();
        mPlatformStrategy.print(mStringToPrint
                                + "(" + loopsDone + ")");
        release();
      }
      mPlatformStrategy.done();
    }
```

Platform-independent print strategy

Implementation of the Ping-Pong Program

- Implementation of the PingPongThread class

```
public class PlayPingPong implements Runnable {
  ...
  static abstract class PingPongThread
                           extends Thread {


    public void run() {
      for (int loopsDone = 1;
           loopsDone <= mMaxIterations;
           ++loopsDone) {
        acquire();
        mPlatformStrategy.print(mStringToPrint
                                + "(" + loopsDone + ")");
        release();
      }
      mPlatformStrategy.done();
    }
```

Scheduling hooks

Implementation of the Ping-Pong Program

- Implementation of the PingPongThread class

```
public class PlayPingPong implements Runnable {
  ...
  static abstract class PingPongThread
                                extends Thread {


    public void run() {
      for (int loopsDone = 1;
           loopsDone <= mMaxIterations;
           ++loopsDone) {
        acquire();
        mPlatformStrategy.print(mStringToPrint
                                + "(" + loopsDone + ")");
        release();
      }
      mPlatformStrategy.done();
    }
```

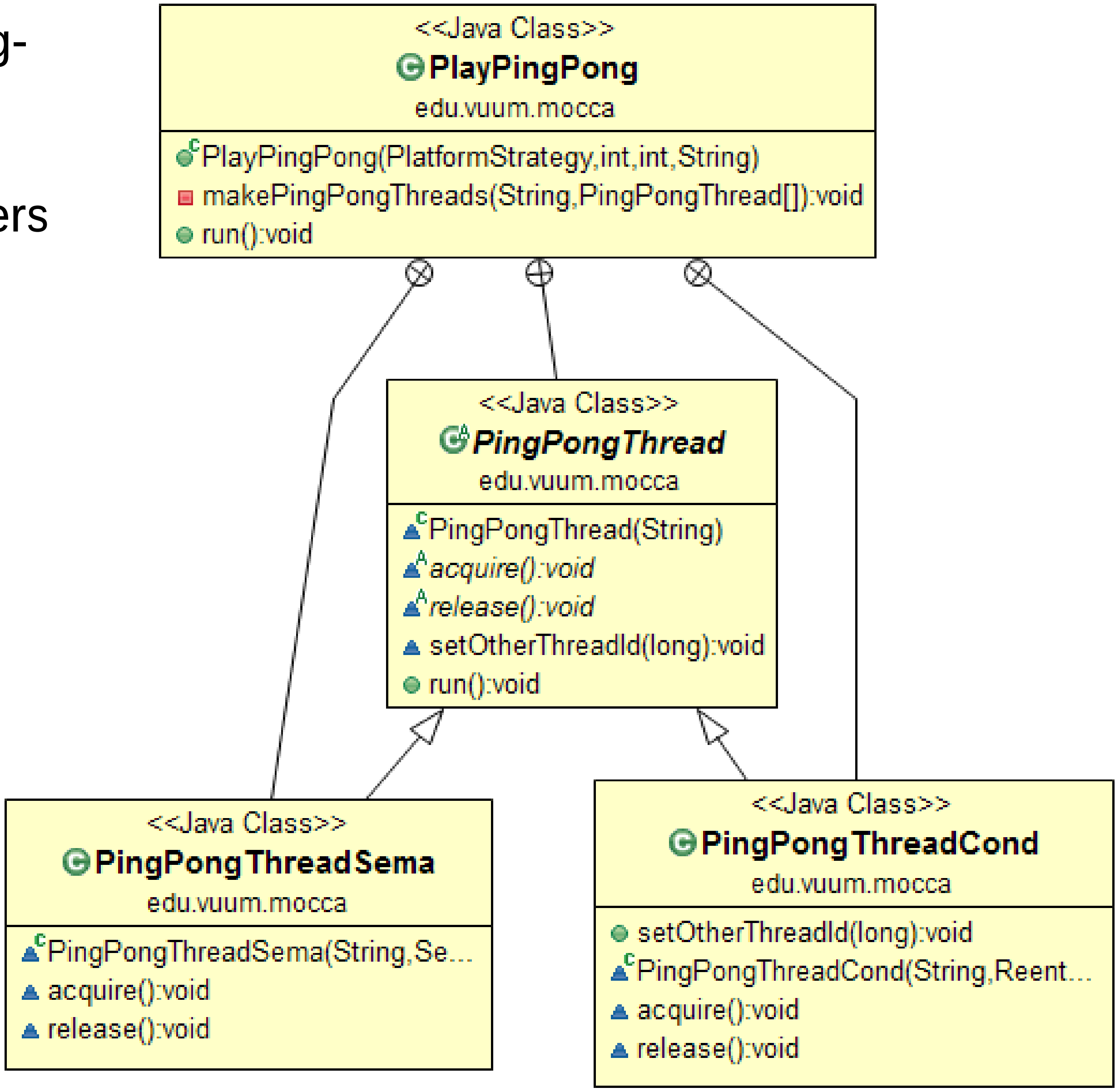
Indicate thread is exiting when loop's done

The underlying CountDownLatch is used to shutdown the PlayPingPong thread

Semaphore Synchronizer Configuration

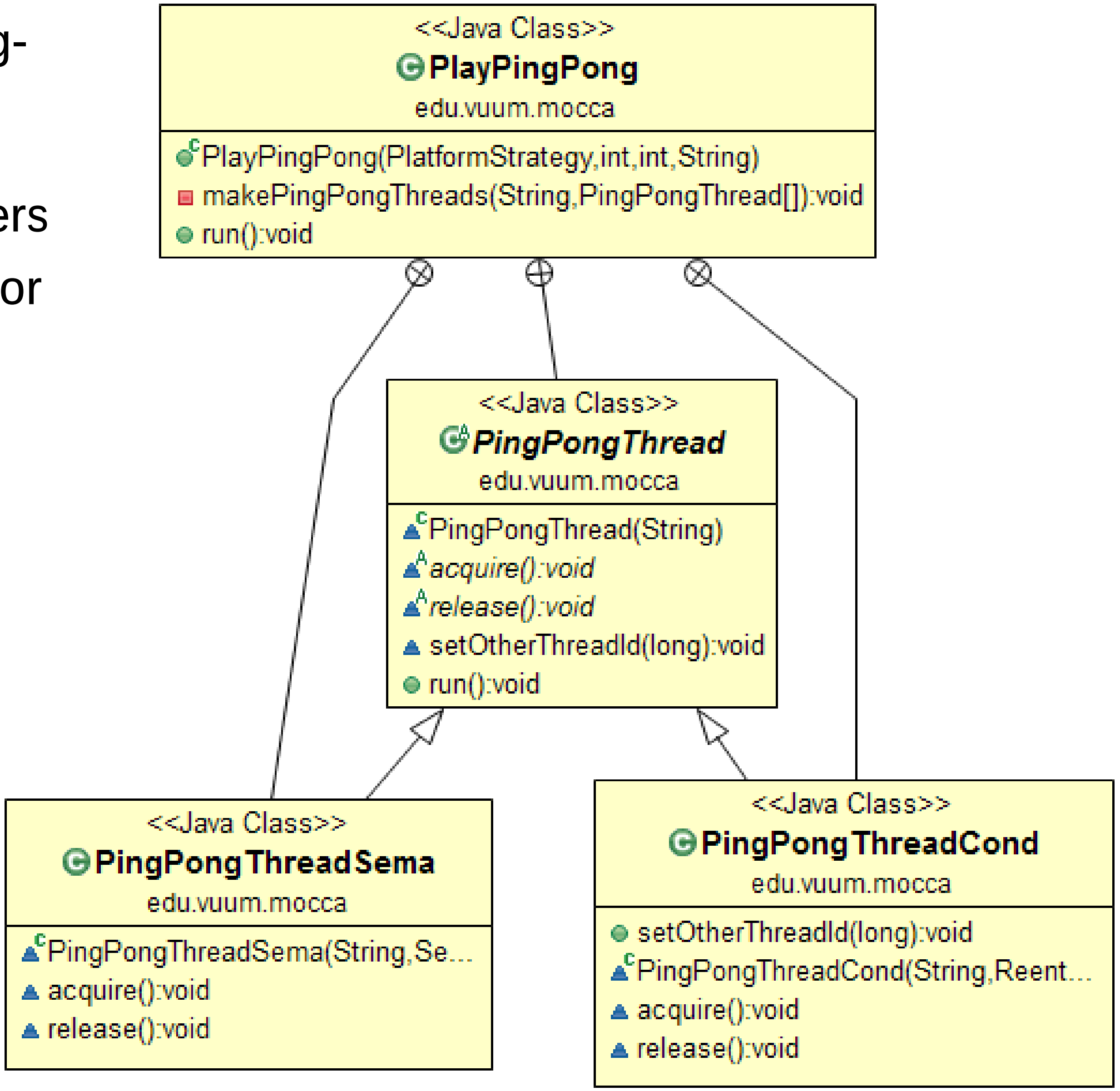
Ping-Pong Program Synchronizers

- The pattern-oriented ping-pong framework can be transparently configured with different synchronizers



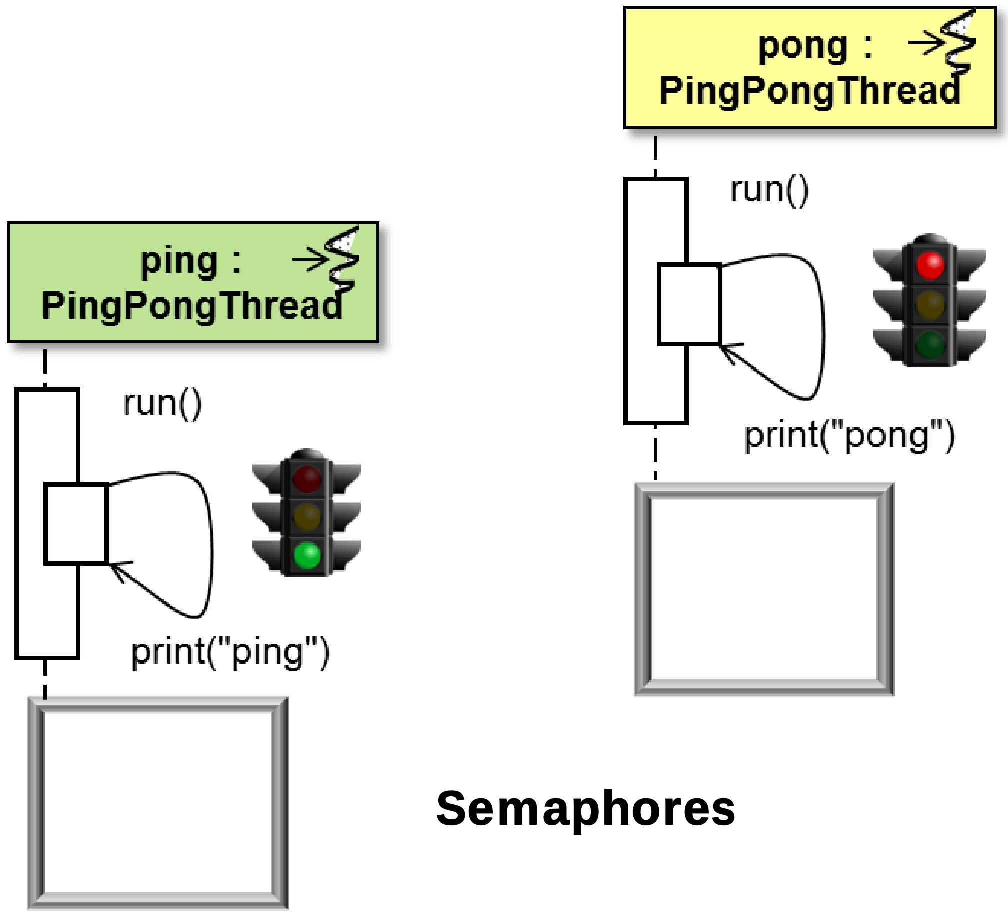
Ping-Pong Program Synchronizers

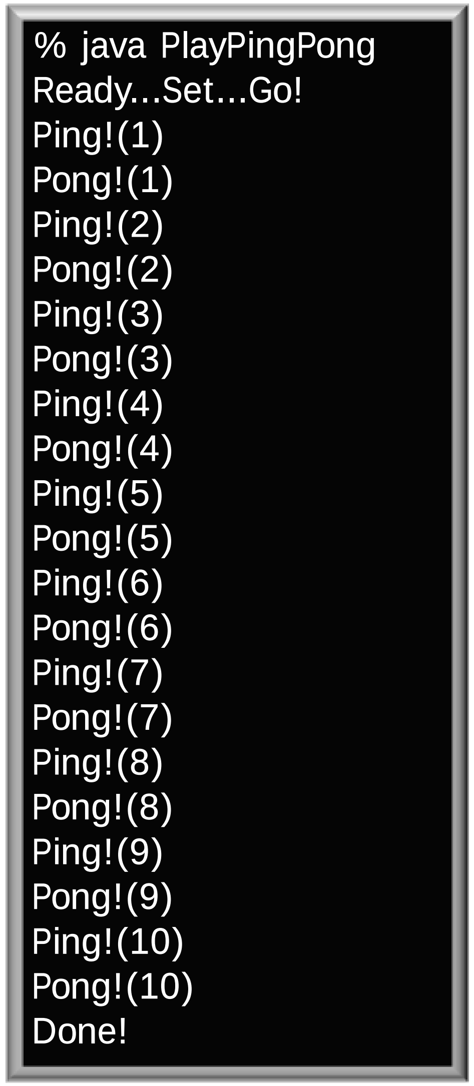
- The pattern-oriented ping-pong framework can be transparently configured with different synchronizers
  - e.g., Java Semaphores or Java ConditionObjects



Semaphore Synchronizer Configuration

- Behavior of the PingPongThreadSema class





```
% java PlayPingPong
Ready...Set...Go!
Ping!(1)
Pong!(1)
Ping!(2)
Pong!(2)
Ping!(3)
Pong!(3)
Ping!(4)
Pong!(4)
Ping!(5)
Pong!(5)
Ping!(6)
Pong!(6)
Ping!(7)
Pong!(7)
Ping!(8)
Pong!(8)
Ping!(9)
Pong!(9)
Ping!(10)
Pong!(10)
Done!
```

Semaphore Synchronizer Configuration

- Implementation of the PingPongThreadSema class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadSema extends PingPongThread {
```

Plays role of "concrete class" in Template Method pattern

```
    private Semaphore mSemas[] = new Semaphore[2];



    PingPongThreadSema(String stringToPrint,
                       Semaphore firstSema, Semaphore secondSema) {
      super(stringToPrint);
      mSemas[FIRST_SEMA] = firstSema;
      mSemas[SECOND_SEMA] = secondSema
    }
```

Semaphore Synchronizer Configuration

- Implementation of the PingPongThreadSema class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadSema extends PingPongThread {



    private Semaphore mSemas[] = new Semaphore[2];
```

Semaphores that schedule the acquire() & release() hook methods

```
    PingPongThreadSema(String stringToPrint,
                       Semaphore firstSema, Semaphore secondSema) {
      super(stringToPrint);
      mSemas[FIRST_SEMA] = firstSema;
      mSemas[SECOND_SEMA] = secondSema
    }
```

Semaphore Synchronizer Configuration

- Implementation of the PingPongThreadSema class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadSema extends PingPongThread {



    private Semaphore mSemas[] = new Semaphore[2];



    PingPongThreadSema(String stringToPrint,
                       Semaphore firstSema, Semaphore secondSema) {
      super(stringToPrint);
      mSemas[FIRST_SEMA] = firstSema;
      mSemas[SECOND_SEMA] = secondSema
    }
```

Pass parameter up to super class & initialize the semaphores

Semaphore Synchronizer Configuration

- Implementation of the PingPongThreadSema class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadSema extends PingPongThread {

    void acquire() {
      mSemas[FIRST_SEMA].acquireUninterruptibly();
    }

    void release() {
      mSemas[SECOND_SEMA].release(); }
  }
```

Hook method for acquiring synchronizer

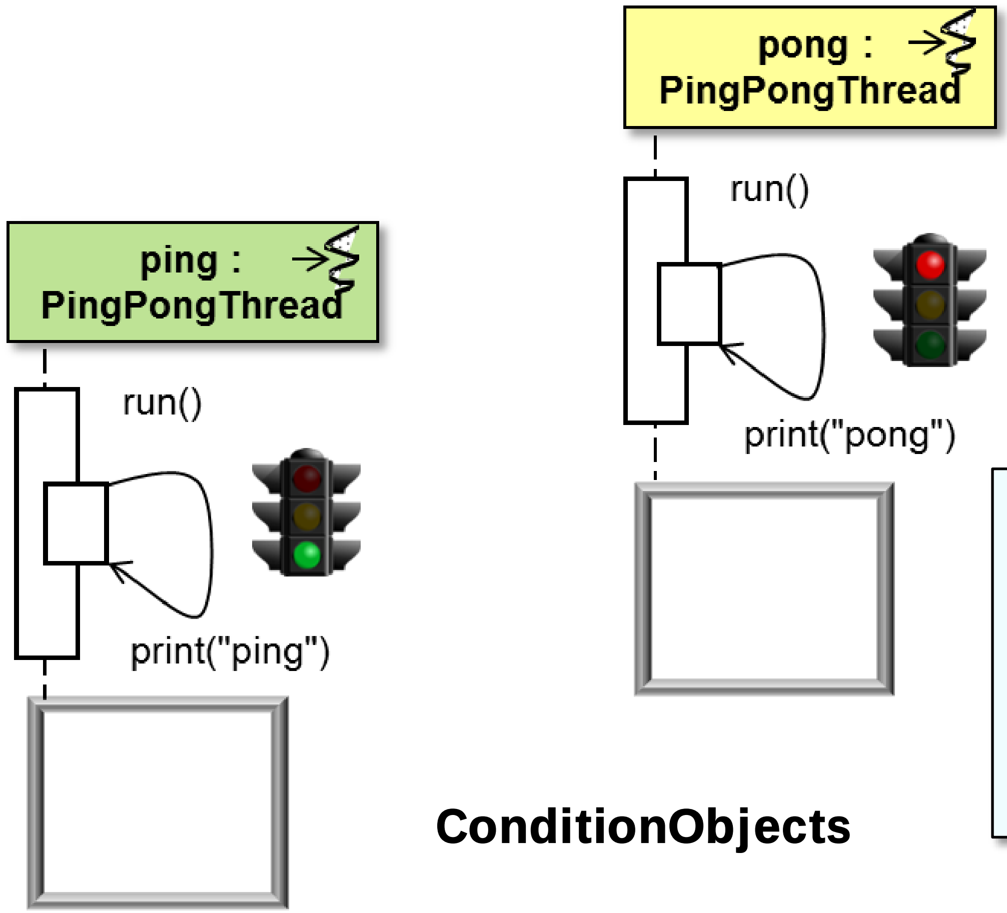
Hook method for releasing synchronizer

acquire()/release() play role of primitive operations in *Template Method* pattern

ConditionObject Synchronizer Configuration

ConditionObject Synchronizer Configuration

- Behavior of the PingPongThreadCond class



Note multiple consecutive printings of "ping" or "pong" per turn

```
% java PlayPingPong
Ready...Set...Go!
Ping!(1)
Ping!(2)
Pong!(1)
Pong!(2)
Ping!(3)
Ping!(4)
Pong!(3)
Pong!(4)
Ping!(5)
Ping!(6)
Pong!(5)
Pong!(6)
Ping!(7)
Ping!(8)
Pong!(7)
Pong!(8)
Ping!(9)
Ping!(10)
Pong!(9)
Pong!(10)
Done!
```

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    private Condition mConds[] = new Condition[2];

    private ReentrantLock mLock = null;

    private int mTurnCountDown = 0;

    public long mOtherThreadId = 0;


    private static long mTurnOwner;
    ...
```

Plays role of "concrete class" in Template Method pattern

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {



    private Condition mConds[] = new Condition[2];

    private ReentrantLock mLock = null;

    private int mTurnCountDown = 0;

    public long mOtherThreadId = 0;


    private static long mTurnOwner;
    ...
```

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {
```

ConditionObjects that schedule the acquire() & release() hook methods

```
    private Condition mConds[] = new Condition[2];

    private ReentrantLock mLock = null;

    private int mTurnCountDown = 0;

    public long mOtherThreadId = 0;


    private static long mTurnOwner;
    ...
```

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {



    private Condition mConds[] = new Condition[2];

    private ReentrantLock mLock = null;

    private int mTurnCountDown = 0;

    public long mOtherThreadId = 0;


    private static long mTurnOwner;
    ...
```

Used together with the ConditionObjects

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {



    private Condition mConds[] = new Condition[2];

    private ReentrantLock mLock = null;

    private int mTurnCountDown = 0;

    public long mOtherThreadId = 0;


    private static long mTurnOwner;
    ...
```

Controls number of consecutive iterations per turn

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {



    private Condition mConds[] = new Condition[2];

    private ReentrantLock mLock = null;

    private int mTurnCountDown = 0;

    public long mOtherThreadId = 0;


    private static long mTurnOwner;
    ...
```

Coordinate the passing of control between ping & pong Threads

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {
    ...
    private final static int FIRST_COND = 0;
    private final static int SECOND_COND = 1;

    PingPongThreadCond(String stringToPrint,
        ReentrantLock lock, Condition firstCond,
        Condition secondCond, boolean isOwner) {
      super(stringToPrint);
      mTurnCountDown = mMaxTurns;
      mLock = lock;
      mConds[FIRST_COND] = firstCond;
      mConds[SECOND_COND] = secondCond;
      if (isOwner) mTurnOwner = this.getId();
      ...
```

Pass parameter up to super class & initialize data members

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void acquire() {
       mLock.lock();

       while (mTurnOwner != this.getId())
          mConds[FIRST_COND].awaitUninterruptibly();

       mLock.unlock();
   }
```

Hook method for acquiring synchronizer

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void acquire() {
       mLock.lock();

       while (mTurnOwner != this.getId())
          mConds[FIRST_COND].awaitUninterruptibly();

       mLock.unlock();
    }
```

Acquire the ReentrantLock

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void acquire() {
       mLock.lock();

       while (mTurnOwner != this.getId())
          mConds[FIRST_COND].awaitUninterruptibly();

       mLock.unlock();
    }
```

Check to see if it’s the turn owner

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void acquire() {
       mLock.lock();

       while (mTurnOwner != this.getId())
          mConds[FIRST_COND].awaitUninterruptibly();

       mLock.unlock();
     }
```

Wait to become the turn owner

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void release() {
       mLock.lock();

       --mTurnCountDown;

       if (mTurnCountDown == 0) {
         mTurnOwner = mOtherThreadId;
         mTurnCountDown = mMaxTurns;
         mConds[SECOND_COND].signal();
       }
       mLock.unlock();
   }
```

Hook method for releasing synchronizer

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void release() {
      mLock.lock();

      --mTurnCountDown;

      if (mTurnCountDown == 0) {
        mTurnOwner = mOtherThreadId;
        mTurnCountDown = mMaxTurns;
        mConds[SECOND_COND].signal();
      }
      mLock.unlock();
    }
```

Acquire the ReentrantLock

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void release() {
       mLock.lock();

       --mTurnCountDown;

       if (mTurnCountDown == 0) {
         mTurnOwner = mOtherThreadId;
         mTurnCountDown = mMaxTurns;
         mConds[SECOND_COND].signal();
       }
       mLock.unlock();
   }
```

Decrement each iteration of our "turn"

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void release() {
       mLock.lock();

       --mTurnCountDown;

       if (mTurnCountDown == 0) {
         mTurnOwner = mOtherThreadId;
         mTurnCountDown = mMaxTurns;
         mConds[SECOND_COND].signal();
       }
       mLock.unlock();
     }
```

Check whether our turn is over

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void release() {
       mLock.lock();

       --mTurnCountDown;

       if (mTurnCountDown == 0) {
         mTurnOwner = mOtherThreadId;
         mTurnCountDown = mMaxTurns;
         mConds[SECOND_COND].signal();
       }
       mLock.unlock();
   }
```

Make the other Thread the turn owner & reset the number of iterations per turn

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void release() {
       mLock.lock();

       --mTurnCountDown;

       if (mTurnCountDown == 0) {
         mTurnOwner = mOtherThreadId;
         mTurnCountDown = mMaxTurns;
         mConds[SECOND_COND].signal();
       }
       mLock.unlock();
     }
```

Signal the other Thread that its turn can begin

ConditionObject Synchronizer Configuration

- Implementation of the PingPongThreadCond class

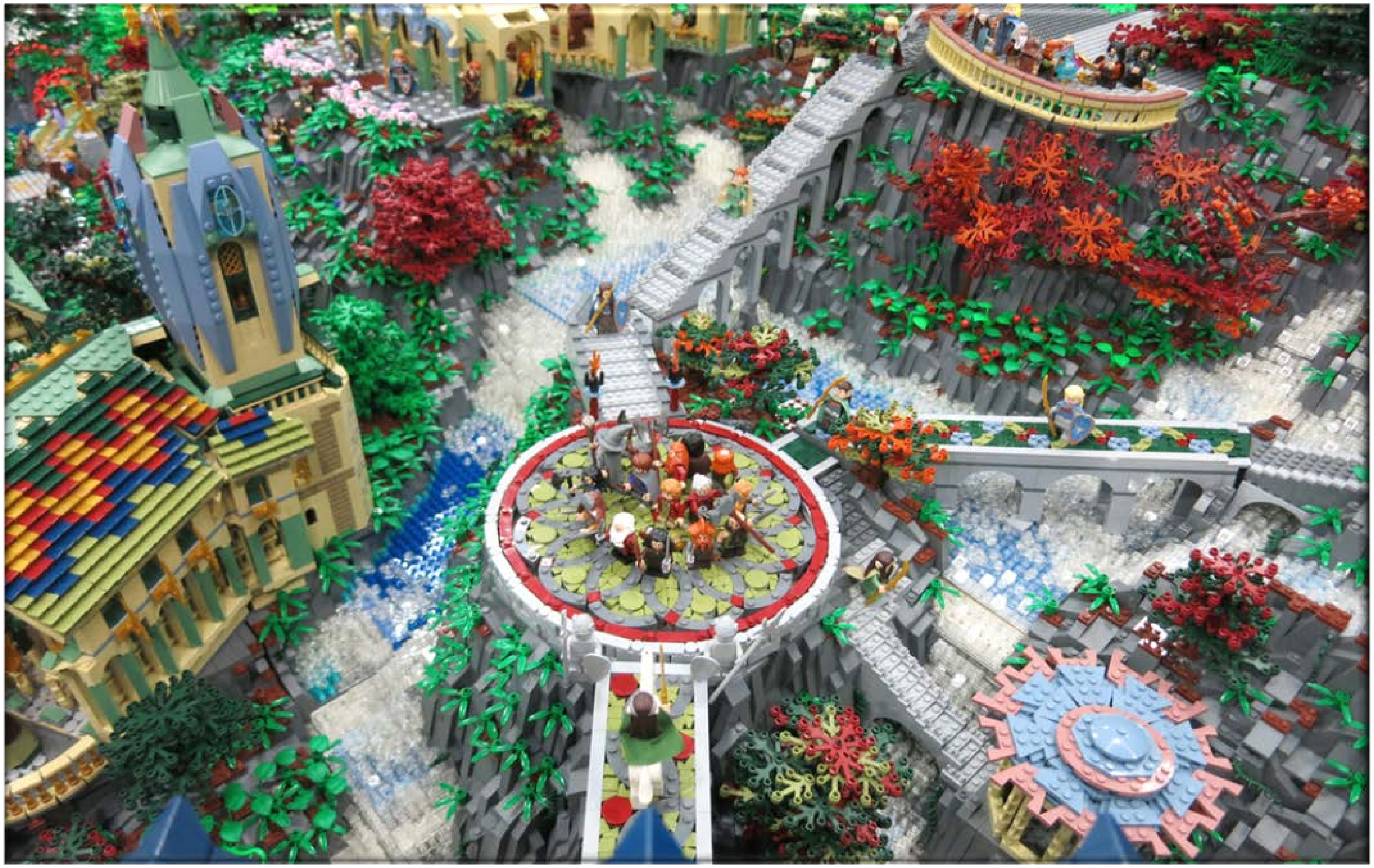
```
public class PlayPingPong implements Runnable {
  ...
  static class PingPongThreadCond extends PingPongThread {

    void release() {
       mLock.lock();

       --mTurnCountDown;

       if (mTurnCountDown == 0) {
         mTurnOwner = mOtherThreadId;
         mTurnCountDown = mMaxTurns;
         mConds[SECOND_COND].signal();
       }
       mLock.unlock();
   }
```

Release the ReentrantLock

Putting All the Pieces Together

Putting All the Pieces Together

Putting All the Pieces Together



Putting All the Pieces Together

- Implementing the PlayPingPong run() method

```
public class PlayPingPong implements Runnable {
  ...
  public void run() {
    mPlatformStrategy.begin();
    ...
    PingPongThread pingPongThreads[] =
      new PingPongThread[2];
    ...

    makePingPongThreads(mSyncMechanism,
                        pingPongThreads);

    pingPongThreads[PING_THREAD].start();
    pingPongThreads[PONG_THREAD].start();

    mPlatformStrategy.awaitDone();
    ...
```

Entry point into PlayPingPong

Putting All the Pieces Together

- Implementing the PlayPingPong run() method

```
public class PlayPingPong implements Runnable {
  ...
  public void run() {
    mPlatformStrategy.begin();
    ...
    PingPongThread pingPongThreads[] =
      new PingPongThread[2];
    ...

    makePingPongThreads(mSyncMechanism,
                        pingPongThreads);

    pingPongThreads[PING_THREAD].start();
    pingPongThreads[PONG_THREAD].start();

    mPlatformStrategy.awaitDone();
    ...
```

Indicate a new game is beginning

Putting All the Pieces Together

- Implementing the PlayPingPong run() method

```
public class PlayPingPong implements Runnable {
  ...
  public void run() {
    mPlatformStrategy.begin();
    ...
    PingPongThread pingPongThreads[] =
      new PingPongThread[2];
    ...

    makePingPongThreads(mSyncMechanism,
                        pingPongThreads);

    pingPongThreads[PING_THREAD].start();
    pingPongThreads[PONG_THREAD].start();

    mPlatformStrategy.awaitDone();
    ...
```

Create array of PingPongThreads

Putting All the Pieces Together

- Implementing the PlayPingPong run() method

```
public class PlayPingPong implements Runnable {
  ...
  public void run() {
    mPlatformStrategy.begin();
    ...
    PingPongThread pingPongThreads[] =
      new PingPongThread[2];
    ...

    makePingPongThreads(mSyncMechanism,
                        pingPongThreads);

    pingPongThreads[PING_THREAD].start();
    pingPongThreads[PONG_THREAD].start();

    mPlatformStrategy.awaitDone();
    ...
```

Use a factory method to initialize Semaphore objects to control thread alternation

See en.wikipedia.org/wiki/Factory_method_pattern for more info

Putting All the Pieces Together

- Implementing the PlayPingPong run() method

```
public class PlayPingPong implements Runnable {
  ...
  public void run() {
    mPlatformStrategy.begin();
    ...
    PingPongThread pingPongThreads[] =
      new PingPongThread[2];
    ...

    makePingPongThreads(mSyncMechanism,
                        pingPongThreads);

    pingPongThreads[PING_THREAD].start();
    pingPongThreads[PONG_THREAD].start();

    mPlatformStrategy.awaitDone();
    ...
```

Start ping & pong threads, which calls their run() methods

Putting All the Pieces Together

- Implementing the PlayPingPong run() method

```
public class PlayPingPong implements Runnable {
  ...
  public void run() {
    mPlatformStrategy.begin();
    ...
    PingPongThread pingPongThreads[] =
      new PingPongThread[2];
    ...

    makePingPongThreads(mSyncMechanism,
                        pingPongThreads);

    pingPongThreads[PING_THREAD].start();
    pingPongThreads[PONG_THREAD].start();

    mPlatformStrategy.awaitDone();
    ...
```

Wait for both threads to exit before exiting run()

Putting All the Pieces Together

- Implementing the PlayPingPong makePingPongThreads() method

Factory method

```
public class PlayPingPong implements Runnable {
  ...
 private void makePingPongThreads(String schedMechanism,
                                   PingPongThread[] pingPongThreads) {
   if (schedMechanism.equals("SEMA")) {
     Semaphore pingSema = new Semaphore(1);
     Semaphore pongSema = new Semaphore(0);



     pingPongThreads[PING_THREAD] =
       new PingPongThreadSema("ping", pingSema, pongSema);
     pingPongThreads[PONG_THREAD] =
       new PingPongThreadSema("pong", pongSema, pingSema);
  }
  ...
```

Putting All the Pieces Together

- Implementing the PlayPingPong makePingPongThreads() method

```
public class PlayPingPong implements Runnable {
  ...
 private void makePingPongThreads(String schedMechanism,
                                  PingPongThread[] pingPongThreads) {
   if (schedMechanism.equals("SEMA")) {
     Semaphore pingSema = new Semaphore(1);
     Semaphore pongSema = new Semaphore(0);


     pingPongThreads[PING_THREAD] =
       new PingPongThreadSema("ping", pingSema, pongSema);
     pingPongThreads[PONG_THREAD] =
       new PingPongThreadSema("pong", pongSema, pingSema);
  }
  ...
```

Select semaphore implementation

Putting All the Pieces Together

- Implementing the PlayPingPong makePingPongThreads() method

```
public class PlayPingPong implements Runnable {
  ...
 private void makePingPongThreads(String schedMechanism,
                                  PingPongThread[] pingPongThreads) {
   if (schedMechanism.equals("SEMA")) {
     Semaphore pingSema = new Semaphore(1);
     Semaphore pongSema = new Semaphore(0);
```

Create two binary semaphores

```
     pingPongThreads[PING_THREAD] =
       new PingPongThreadSema("ping", pingSema, pongSema);
     pingPongThreads[PONG_THREAD] =
       new PingPongThreadSema("pong", pongSema, pingSema);
  }
  ...
```

Putting All the Pieces Together

- Implementing the PlayPingPong makePingPongThreads() method

```
public class PlayPingPong implements Runnable {
    ...
  private void makePingPongThreads(String schedMechanism,
                                   PingPongThread[] pingPongThreads) {
    if (schedMechanism.equals("SEMA")) {
      Semaphore pingSema = new Semaphore(1);
      Semaphore pongSema = new Semaphore(0);


      pingPongThreads[PING_THREAD] =
        new PingPongThreadSema("ping", pingSema, pongSema);
      pingPongThreads[PONG_THREAD] =
        new PingPongThreadSema("pong", pongSema, pingSema);
    }
    ...
```

Begin unlocked

Putting All the Pieces Together

- Implementing the PlayPingPong makePingPongThreads() method

```
public class PlayPingPong implements Runnable {
  ...
 private void makePingPongThreads(String schedMechanism,
                           PingPongThread[] pingPongThreads) {
   if (schedMechanism.equals("SEMA")) {
     Semaphore pingSema = new Semaphore(1);
     Semaphore pongSema = new Semaphore(0);



     pingPongThreads[PING_THREAD] =
       new PingPongThreadSema("ping", pingSema, pongSema);
     pingPongThreads[PONG_THREAD] =
       new PingPongThreadSema("pong", pongSema, pingSema);
  }
  ...
```

Create two thread objects initialized for “ping” & “pong”

Putting All the Pieces Together

- Implementing the PlayPingPong makePingPongThreads() method

```
public class PlayPingPong implements Runnable {
  ...
 private void makePingPongThreads(String schedMechanism,
                            PingPongThread[] pingPongThreads) {
   if (schedMechanism.equals("SEMA")) {
     Semaphore pingSema = new Semaphore(1);
     Semaphore pongSema = new Semaphore(0);




     pingPongThreads[PING_THREAD] =
       new PingPongThreadSema("ping", pingSema, pongSema);
     pingPongThreads[PONG_THREAD] =
       new PingPongThreadSema("pong", pongSema, pingSema);
  }
  ...
```

Create two thread objects initialized for “ping” & “pong”

Putting All the Pieces Together

- Implementing the PlayPingPong makePingPongThreads() method

```
public class PlayPingPong implements Runnable {
   ...
  private void makePingPongThreads(String schedMechanism,
                                   PingPongThread[] pingPongThreads) {
    if (schedMechanism.equals("SEMA")) {
      Semaphore pingSema = new Semaphore(1);
      Semaphore pongSema = new Semaphore(0);



      pingPongThreads[PING_THREAD] =
        new PingPongThreadSema("ping", pingSema, pongSema);
      pingPongThreads[PONG_THREAD] =
        new PingPongThreadSema("pong", pongSema, pingSema);
   }
   ...
```

Create two thread objects initialized for “ping” & “pong”

Putting All the Pieces Together

- Implementation of the Main class

main() entry point into the ping/pong program

```
public class Main {
  ...
  public void main(String[] args) {
    PlatformStrategy.instance
        (new PlatformStrategyFactory
           (System.out, null).makePlatformStrategy());

    Options.instance().parseArgs(args);

    PlayPingPong playPingPong =
      new PlayPingPong(PlatformStrategy.instance(),
                       Options.instance().maxIterations(),
                       Options.instance().maxTurns(),
                       Options.instance().syncMechanism());

    new Thread (playPingPong).start();
    ...
```

Putting All the Pieces Together

- Implementation of the Main class

Initialize PlatformStrategy to ConsolePlatformStrategy

```
public class Main {
  ...
  public void main(String[] args) {
    PlatformStrategy.instance
       (new PlatformStrategyFactory
          (System.out, null).makePlatformStrategy());

    Options.instance().parseArgs(args);

    PlayPingPong playPingPong =
      new PlayPingPong(PlatformStrategy.instance(),
                       Options.instance().maxIterations(),
                       Options.instance().maxTurns(),
                       Options.instance().syncMechanism());

    new Thread (playPingPong).start();
    ...
```

Putting All the Pieces Together

- Implementation of the Main class

```
public class Main {
  ...
  public void main(String[] args) {
    PlatformStrategy.instance
       (new PlatformStrategyFactory
          (System.out, null).makePlatformStrategy());

    Options.instance().parseArgs(args);

    PlayPingPong playPingPong =
      new PlayPingPong(PlatformStrategy.instance(),
                       Options.instance().maxIterations(),
                       Options.instance().maxTurns(),
                       Options.instance().syncMechanism());

    new Thread (playPingPong).start();
    ...
```

Initialize the Options

Putting All the Pieces Together

- Implementation of the Main class

```
public class Main {
  ...
  public void main(String[] args) {
    PlatformStrategy.instance
       (new PlatformStrategyFactory
          (System.out, null).makePlatformStrategy());

    Options.instance().parseArgs(args);

    PlayPingPong playPingPong =
      new PlayPingPong(PlatformStrategy.instance(),
                       Options.instance().maxIterations(),
                       Options.instance().maxTurns(),
                       Options.instance().syncMechanism());

    new Thread (playPingPong).start();
    ...
```

Initialize PlayPingPong object with designated parameters

Putting All the Pieces Together

- Implementation of the Main class

```
public class Main {
  ...
  public void main(String[] args) {
    PlatformStrategy.instance
       (new PlatformStrategyFactory
          (System.out, null).makePlatformStrategy());

    Options.instance().parseArgs(args);

    PlayPingPong playPingPong =
      new PlayPingPong(PlatformStrategy.instance(),
                       Options.instance().maxIterations(),
                       Options.instance().maxTurns(),
                       Options.instance().syncMechanism());

    new Thread (playPingPong).start();
    ...
```

Start a thread to play ping pong concurrently!

Putting All the Pieces Together

- Implementation of the Main class

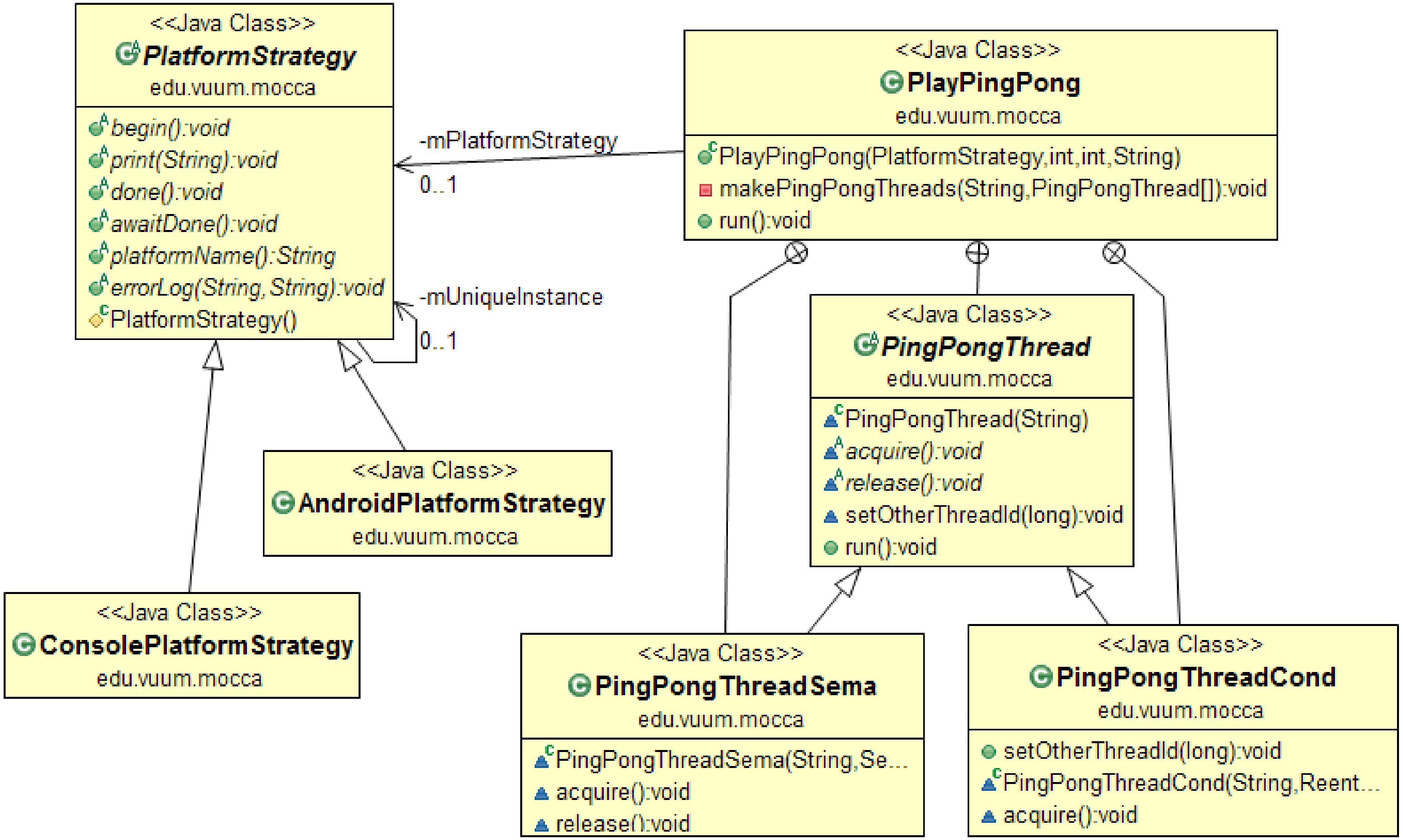
```
public class Main {
  ...
  public void main(String[] args) {
    PlatformStrategy.instance
       (new PlatformStrategyFactory
          (System.out, null).makePlatformStrategy());

    Options.instance().parseArgs(args);

    PlayPingPong playPingPong =
      new PlayPingPong(PlatformStrategy.instance(),
                       Options.instance().maxIterations(),
                       Options.instance().maxTurns(),
                       Options.instance().syncMechanism());

    new Thread (playPingPong).start();
    ...
```

Start a thread to play ping pong concurrently!

Summary

Summary

- Pattern-oriented framework implements a concurrent ping/pong program



Summary

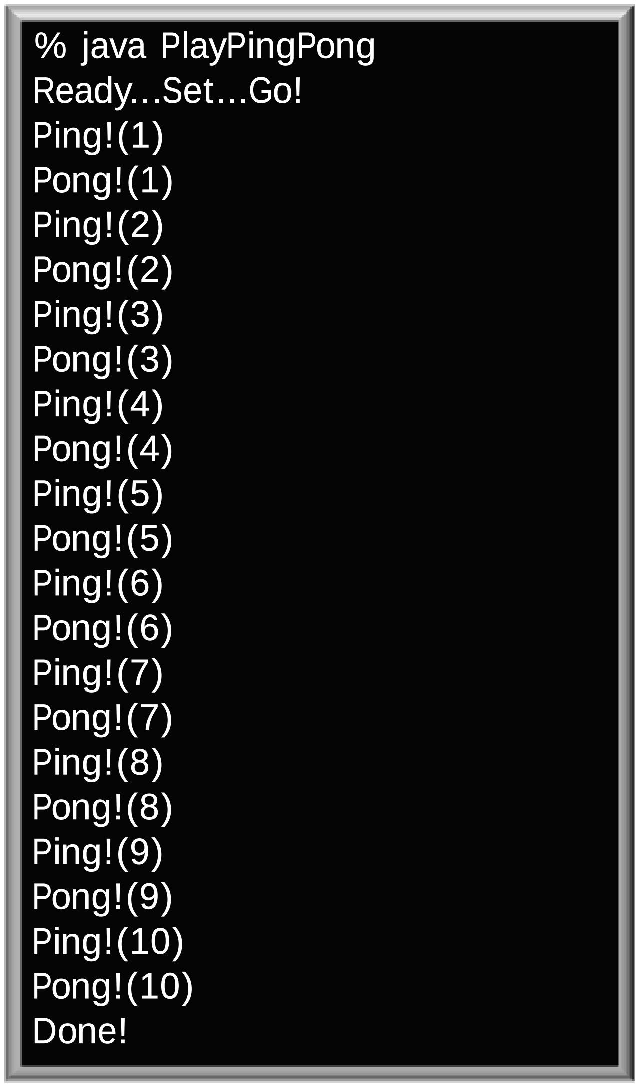
- Pattern-oriented framework implements a concurrent ping/pong program
  - Fixes problems with the PingPongWrong implementation

```
% java PingPongWrong
Ready...Set...Go!
Ping!(1)
Ping!(2)
Ping!(3)
Ping!(4)
Ping!(5)
Ping!(6)
Ping!(7)
Ping!(8)
Ping!(9)
Ping!(10)
Pong!(1)
Pong!(2)
Pong!(3)
Pong!(4)
Pong!(5)
Pong!(6)
Pong!(7)
Pong!(8)
Pong!(9)
Pong!(10)
Done!
```

See earlier part on “Motivating Java Synchronization & Scheduling Mechanisms

Summary

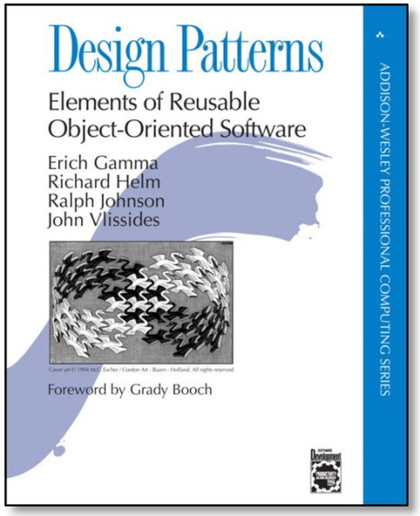
- Pattern-oriented framework implements a concurrent ping/pong program
  - Fixes problems with the PingPongWrong implementation
  - Output correctly alternates printing "ping" & "pong" to output since Java Semaphores, ConditionObjects, & CountDownLatches are used

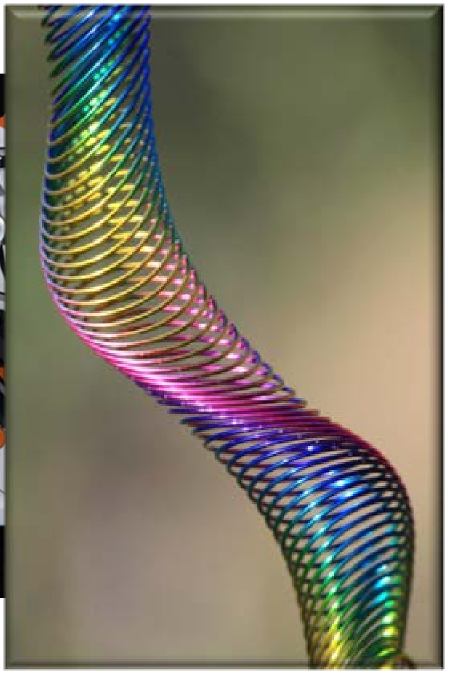


```
% java PlayPingPong
Ready...Set...Go!
Ping!(1)
Pong!(1)
Ping!(2)
Pong!(2)
Ping!(3)
Pong!(3)
Ping!(4)
Pong!(4)
Ping!(5)
Pong!(5)
Ping!(6)
Pong!(6)
Ping!(7)
Pong!(7)
Ping!(8)
Pong!(8)
Ping!(9)
Pong!(9)
Ping!(10)
Pong!(10)
Done!
```

Summary

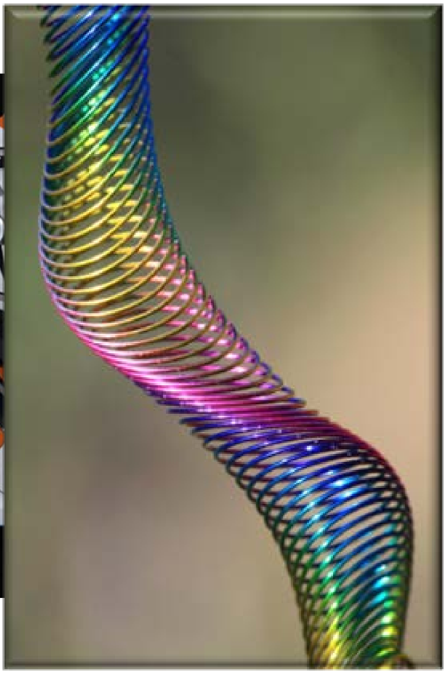
- Pattern-oriented framework implements a concurrent ping/pong program
- GoF patterns provide several benefits
  - *Template Method & Factory Method* simplify systematic reuse & flexibility


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



Summary

- Pattern-oriented framework implements a concurrent ping/pong program
- GoF patterns provide several benefits
  - *Template Method & Factory Method* simplify systematic reuse & flexibility



```
% java PlayPingPong
Ready...Set...Go!
Ping!(1)
Ping!(1)
Pong!(1)
Pong!(1)
Ping!(2)
Ping!(2)
Pong!(2)
Pong!(2)
Ping!(3)
Ping!(3)
Pong!(3)
Pong!(3)
Ping!(4)
Ping!(4)
Pong!(4)
Pong!(4)
Ping!(5)
Ping!(5)
Pong!(5)
Pong!(5)
Done!
```

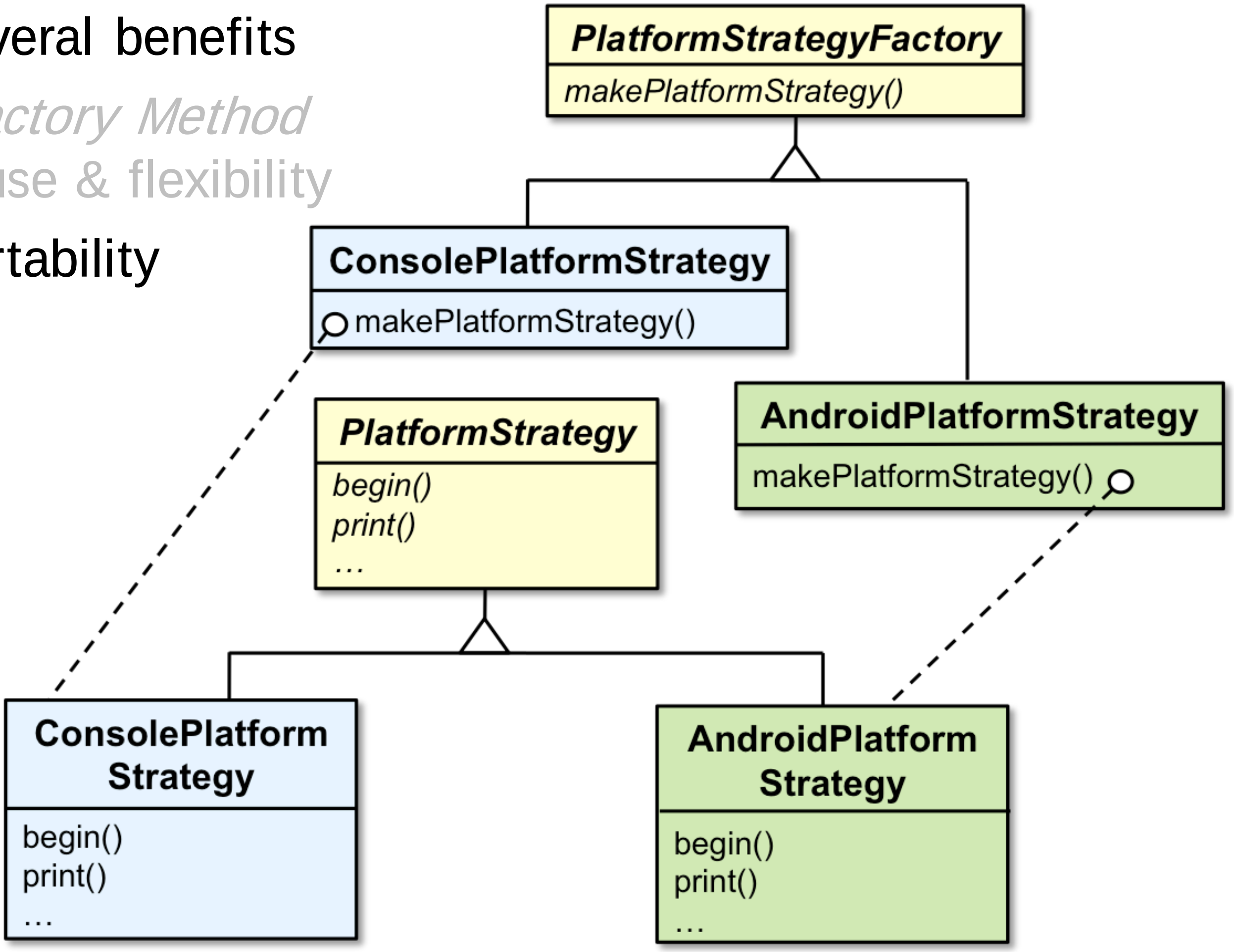
Summary

- Pattern-oriented framework implements a concurrent ping/pong program
- GoF patterns provide several benefits
  - *Template Method & Factory Method* simplify systematic reuse & flexibility
  - *Strategy* enhanced portability

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

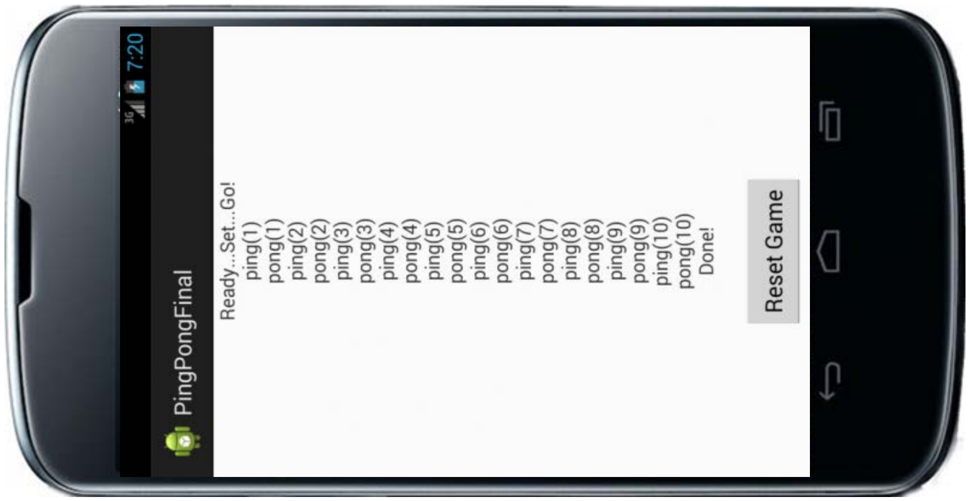
Summary

- Pattern-oriented framework implements a concurrent ping/pong program
- GoF patterns provide several benefits
  - *Template Method & Factory Method* simplify systematic reuse & flexibility
  - *Strategy* enhanced portability



Summary

- Pattern-oriented framework implements a concurrent ping/pong program
- GoF patterns provide several benefits
  - *Template Method* & *Factory Method* simplify systematic reuse & flexibility
  - *Strategy* enhanced portability

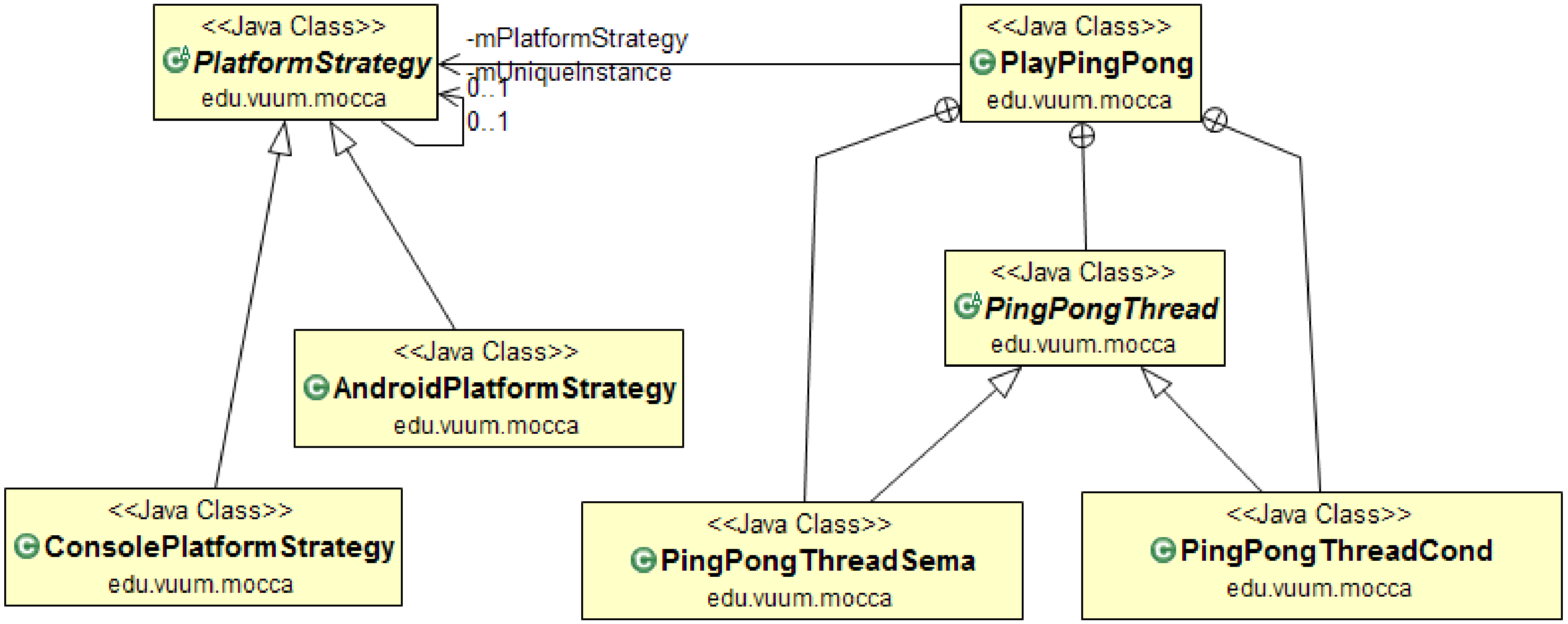


```
Ready...Set...Go!
ping(1)
pong(1)
ping(2)
pong(2)
ping(3)
pong(3)
ping(4)
pong(4)
ping(5)
pong(5)
ping(6)
pong(6)
ping(7)
pong(7)
ping(8)
pong(8)
ping(9)
pong(9)
ping(10)
pong(10)
Done!
```

```
% java PlayPingPong
Ready...Set...Go!
Ping!(1)
Ping!(1)
Pong!(1)
Pong!(1)
Ping!(2)
Ping!(2)
Pong!(2)
Pong!(2)
Ping!(3)
Ping!(3)
Pong!(3)
Pong!(3)
Ping!(4)
Ping!(4)
Pong!(4)
Pong!(4)
Ping!(5)
Ping!(5)
Pong!(5)
Pong!(5)
Done!
```

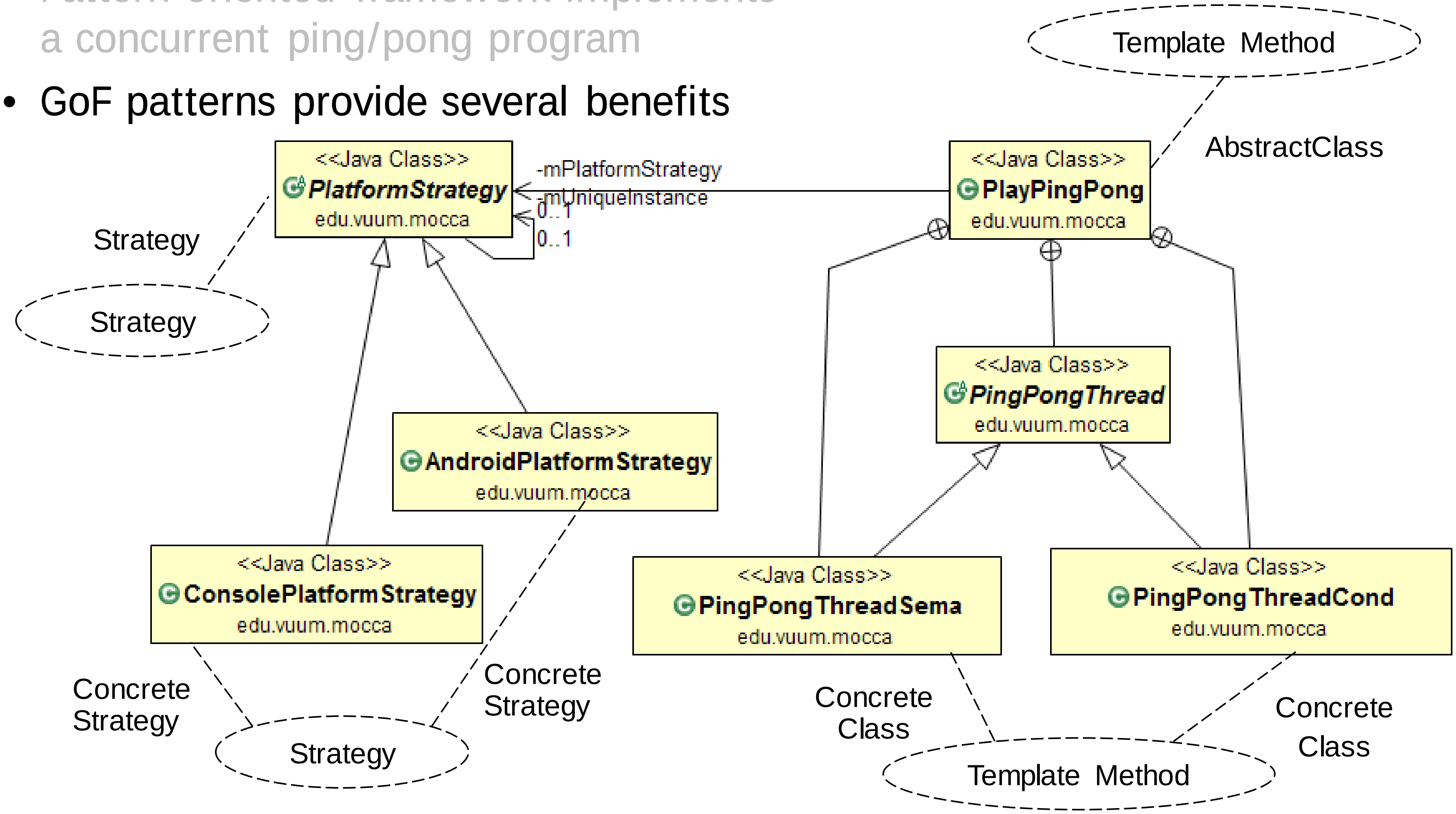
Summary

- Pattern-oriented framework implements a concurrent ping/pong program
- GoF patterns provide several benefits



Summary

- Pattern-oriented framework implements a concurrent ping/pong program
- GoF patterns provide several benefits



Patterns simplify understanding & extension of program structure & functionality