class Firemen
{
    public static final int MIN_X = 1;
    public static final int MAX_X = 10;
    private int x_;

    // @INV: MIN_X <= x_ <= MAX_X
    Firemen() { x_ = 5; }

    public synchronized void repaint()
    {
        System.out.println(x_);
    }

    public synchronized void moveLeft()
    {
        if (x_ > MIN_X)
            x_--;
        repaint();
    }
}
```java
public synchronized void moveRight()
{
    if (x_ < MAX_X)
        x_++;
    repaint();
}

public synchronized int getX() { return x_; }
```

class ClicksList
{
  private LinkedList list_
;

  // add item to the end of the list
  public synchronized void add(String s) throws InterruptedException
  {
      list_.add(s);
      notifyAll();
  }

  // remove the first string in the list if equals to given parameter s
  public synchronized void get(String s) throws InterruptedException
  {
      while (list_.size() == 0 ||
             ! list_.getFirst().equals(s))
          wait();
      list_.removeFirst(); // Remove first element of list
      notifyAll();
  }
}

class UserInterface extends Thread
{
  private ClicksList clicksList_
;

  UserInterface (ClicksList clicksList) {
      clicksList_ = clicksList;
  }

  public void onClickLeft() {
      try {
          clicksList_.add("L");
      } catch (InterruptedException e) {
      }
  }
}
```
public void onClickRight() {
    try {
        clicksList_.add("R");
    } catch (InterruptedException e) {
    }
}

public void run() {
    // Get events from UI and dispatch to onClickLeft/Right
}

class Left extends Thread {
    private ClicksList clicksList_;
    private Firemen firemen_; 
    
    Left(ClicksList clicksList, Firemen firemen) {
        firemen_ = firemen; clicksList_ = clicksList;
    }
    
    public void run() {
        while (true) {
            try{
                clicksList_.get("L");
                firemen_.moveLeft();
            } catch (InterruptedException e) {
            }
        }
    }
}

class Right extends Thread {
    private ClicksList clicksList_; 
    private Firemen firemen_; 
    
    Right(ClicksList clicksList, Firemen firemen) {
        firemen_ = firemen; clicksList_ = clicksList;
    }
    
    public void run() {
        while (true) {
            try{
                clicksList_.get("R");
                firemen_.moveRight();
            } catch (InterruptedException e) {
            }
        }
    }
}

class UserInterface extends Thread {
    private ClicksCounter clicksCounter_; 
    UserInterface (ClicksCounter  clicksCounter) {
        clicksCounter_ = clicksCounter;
    }
    
    public void onClickLeft() {
    }
    
    public void run() {
        // Get events from UI and dispatch to onClickLeft/Right
    }
}
try {
    clicksCounter_.incLeft();
} catch (InterruptedException e) {
}

public void onClickRight() {
    try {
        clicksCounter_.incRight();
    } catch (InterruptedException e) {
    }
}

class ClicksCounter {
    private int leftNum_, rightNum_;  
    ClicksCounter() { leftNum_ = 0; rightNum_ = 0; }

    public synchronized void incLeft() throws InterruptedException {
        leftNum_++;
        notifyAll();
    }

    public synchronized void incRight() throws InterruptedException {
        rightNum_++;
        notifyAll();
    }

    public synchronized int resetLeft() throws InterruptedException {
        while (leftNum_ == 0)
            wait();
        int ret = leftNum_;  
        leftNum_ = 0;
        return ret;
    }

    public synchronized int resetRight() throws InterruptedException {
        while (rightNum_ == 0)
            wait();
        int ret = rightNum_;  
        rightNum_ = 0;
        return ret;
    }
}

class Left extends Thread {
    private ClicksCounter clicksCounter_;  
    private Firemen firemen_;  

    Left(ClicksCounter clicksCounter, Firemen firemen) {
        firemen_ = firemen;
        clicksCounter_ = clicksCounter;
    }

    public void run() {
        while (true) {
            try {
                int count = clicksCounter_.resetLeft();
            }
        }
    }
}
```java
class Right extends Thread {
    private ClicksCounter clicksCounter_;
    private Firemen firemen_;

    Right(ClicksCounter clicksCounter, Firemen firemen) {
        firemen_ = firemen;
        clicksCounter_ = clicksCounter;
    }

    public void run() {
        while (true) {
            try {
                int count = clicksCounter_.resetRight();
                for (int i = 0; i < count; i++)
                    firemen_.moveRight();
            } catch (InterruptedException e) {
            }
        }
    }
}
```

```java
class Victim {
    private final int x_; // Position from which the victim is thrown
    private int y_; // Vertical position of the victim in [5,1]
    // @inv: 1 <= y_ <= 5 && MIN_X <= x_ <= MAX_X
    public Victim(int x) {
        x_ = x;
        y_ = 5;
    }

    public int getY() {
        return y_; // Vertical position of the victim
    }
}
```
class Thrower extends Thread {

private Victim v_;  
private Firemen f_;  
private int score_;  
public Thrower(Firemen f) {
    f_ = f;  
v_ = null;  
score_ = 0;  
}

public synchronized int getScore() { return score_; }  
public Victim getVictim() { return v_; }  
public void run() {
    try{  
        Random random = new Random();  
        while (true) {
            if (v_ == null) {  
                // select a random number in range [min/max]
                int initX = random.nextInt(Firemen.MAX_X - Firemen.MIN_X) + Firemen.MIN_X;  
v_ = new Victim(initX);  
            }
            v_.fall();  
sleep(500);  
if (v_.getX() == 1) {
    synchronized(v_) {  
        synchronized(f_) {  
        if (f_.getX() == v_.getX()) {
            score_++;  
        } else {
            score_--;  
        }
    v_ = null;  
}}
    }
  }
} catch (InterruptedException e) {
}
}

public int getX() {
    return x_;  
}

public void fall() {
if (y_ > 1)
    y_--;  
}

}
```cpp
class X {
public:
    X( const C& );
    D Function1( int, char* );
    D Function1( int, C );
    B& Function2( B );
private:
    std::string  name_
    std::list<C> clist_
    D d_
};
```

```cpp
#include "b.h" // class B
#include "c.h" // class C
#include "d.h" // class D
#include <iostream>
#include <list>
#include <string>
class X {
public:
    X( const C& );
    D Function1( int, char* );
    D Function1( int, C );
    B& Function2( B );
private:
    std::string  name_; 
    std::list<C> clist_; 
    D d_; 
};
```

```cpp
class Person {
public:
    Person(std::string& name, long id) : name_(name), id_(id) {}
    ~Person() {}
private:
    std::string& name_; 
    long id_; 
};
```

```cpp
class Student : public Person {
public:
    Student(std::string& name, long id) : Person(name, id) {}
    Student(std::string& name, long id, const std::vector<int*>& courses) :
```
Person(name, id), courses_(courses) {}
virtual ~Student() {
    std::vector<int*>::iterator it;
    for (it = courses_.begin(); it != courses_.end(); ++it)
        delete (*it);
}
const std::vector<int*>& getCourses() const { return courses_; }
private:
    std::vector<int*> courses_;
Remote ObjectHost
sockets
TCP

TCP socket

UDP checksum

multicast

Host A

Host B

Host C

threads pool

reactors

data

reactions

ports

reactor

Thread Pool

Reactor
Thread Pool and Reactor Host: transfers are initiated by the thread pool, and the Reactors are executed on a total of 33 threads. Each thread executes a Reactor on a total of 99 threads. The Reactors are executed in pairs and each pair executes a Reactor on a total of 12 threads.

Shall we ask the following questions to the SQL database?

1. How many words are in the database?
2. How many analyses are in the database?
3. What is the gender and number of the subject?
4. What is the primary key of the database?

Words:

<table>
<thead>
<tr>
<th>Primary Key</th>
<th>Str</th>
<th>Freq</th>
</tr>
</thead>
</table>

Analyses:

<table>
<thead>
<tr>
<th>Primary Key</th>
<th>ID</th>
<th>POS</th>
<th>Gender</th>
<th>Number</th>
</tr>
</thead>
</table>

Words Analyses:

<table>
<thead>
<tr>
<th>Primary Key</th>
<th>WordStr</th>
<th>AnalysisId</th>
</tr>
</thead>
</table>

An analyses is defined as an analysisId, and a word is defined as a wordStr. The following is a list of the analyses:

- "gender"
- "POS"
- "number"
- "wordStr"

The gender and number are defined as integers, and the wordStr is defined as a string.

The SQL database contains the following tables:

- Words: containing the primary key, str, and freq.
- Analyses: containing the primary key, id, pos, gender, and number.
- Words Analyses: containing the primary key, wordStr, and analysisId.