Concurrent Object Oriented Languages

java.util.concurrent.atomic

wiki.eecs.yorku.ca/course/6490A

Sieve of Eratothenes

```
public class Generator implements CSProcess
{
   private ChannelOutputInt out;

   public Generator(ChannelOutputInt out)
   {
      this.out = out;
   }

   public void run()
   {
       for (int i = 2; true; i++)
       {
           this.out.write(i);
       }
   }
}
```

Sieve of Eratothenes

```
public class Absorberator implements CSProcess
{
   private ChannelInputInt in;

   public Absorberator(ChannelInputInt in)
   {
      this.in = in;
   }

   public void run()
   {
      while (true)
      {
         this.in.read();
      }
  }
}
```

Sieve of Eratothenes

```
public class Sieve implements CSProcess
{
   private ChannelInputInt in;
   private ChannelOutputInt out;

   public Sieve(ChannelInputInt in,
                ChannelOutputInt out)
   {
      this.in = in;
      this.out = out;
   }
```

Sieve of Eratothenes

```
   public void run()
   {
      int prime = this.in.read();
      while (true)
      {
         int i = this.in.read();
         if (i % prime != 0)
         {
            this.out.write(i);
         }
      }
   }
}
```

Sieve of Eratothenes

```
public class Eratosthenes
{
   public static void main (String[] args)
   {
      final int NUMBER = Integer.parseInt(args[0]);
      One2OneChannelInt[] channel =
        new One2OneChannelInt[NUMBER + 1];
      for (int i = 0; i <= NUMBER; i++)
      {
         channel[i] = Channel.one2oneInt();
      }
```

Sieve of Eratothenes

```
   CSProcess[] process = new CSProcess[NUMBER + 2];
   process[0] = new Generator(channel[0].out());
   for (int i = 1; i <= NUMBER; i++)
   {
      process[i] =
        new Sieve(channel[i - 1].in(),
                  channel[i].out());
   }
   process[NUMBER + 1] =
      new Absorberator(channel[NUMBER].in());
   new Parallel(process).run();
}
```

Concurrent ML

Concurrent ML (CML for short) is a concurrent extension of the Standard ML programming language. CML was designed by John Reppy, who is a professor at the University of Chicago.

CML is not as verbose as Java.

Sieve of Eratothenes

```
fun generator() =
let
  val out = channel()
  fun loop(n) = (send(out, n); loop(n+1))
in
  spawn(fn () => loop(2));
  out
end
```

Sieve of Eratothenes

```
fun filter(p, input) =
let
  val out = channel()
  fun loop() =
  let
    val i = recv(input)
  in
    (if i mod p <> 0
     then send(out, i) else ());
    loop()
  end
in
  spawn loop; out
end
```

Sieve of Eratothenes

```
fun sieve() =
let
  val primes = channel()
  fun head(input) =
  let
    val p = recv(input)
  in
    (send(primes, p);
    head(filter(p, input)))
  end
in
  spawn (fn () => head(generator())); primes
end
```

java.util.concurrent.atomic

The Java package java.util.concurrent.atomic contains classes that support lock-free thread-safe programming on single variables.

AtomicReference<V>

Objects of type AtomicReference<V> contain a value of type V that may be updated atomically.

The class contains the method

```
public final boolean compareAndSet(V expect,
                                   V update)
```

It atomically sets the value to `update` if the current value of the object == `expect`. It returns true if the update is successful, and false otherwise.

AtomicReference<V>

Problem

Implement a Stack by means of AtomicReference<V>.

Problem

Implement a Node class.

Stack

```
pop():
success = false;
while not success do
  node = top
  if (node == null) throw an exception
  success = CAS(top, node, node.getNext());
return element of node
```

Stack

```
push(element):
node = node with element
success = false;
while not success do
  node.next = top
  succes = CAS(top, node.getNext(), node)
```

AtomicReferenceFieldUpdater<T,V>

The class contains the method

```
public static <U,W> AtomicReferenceFieldUpdater<U,W
   newUpdater(Class<U> tclass,
              Class<W> vclass,
              String fieldName)
```

It returns an object that can be used to atomically update the field with the given `fieldName`.

AtomicReferenceFieldUpdater<T,V>

The class contains the method

```
public abstract boolean compareAndSet(T object,
                                      V expect,
                                      V update)
```

It atomically sets the field of the given `object` managed by this updater to the given `update` value if the current value === `expect`.

This method is guaranteed to be atomic with respect to other calls to compareAndSet, but not necessarily with respect to other changes in the field.

AtomicReferenceFieldUpdater<T,V>

Problem

Implement a Stack by means of AtomicReferenceFieldUpdater<T,V>.

AtomicInteger

The class contains methods such as

```
public final int incrementAndGet()
```

and

```
public final int getAndAdd(int delta)
```

AtomicStampedReference<V>

This class is similar to AtomicReference<V> but not only manipulates an object but also an integer stamp. The class contains the method

```
public boolean compareAndSet(V expectedReference,
                             V newReference,
                             int expectedStamp,
                             int newStamp)
```