Object – Oriented Technology

Chapter 5: Concurrent Models

Suad Alagić

Springer 2016

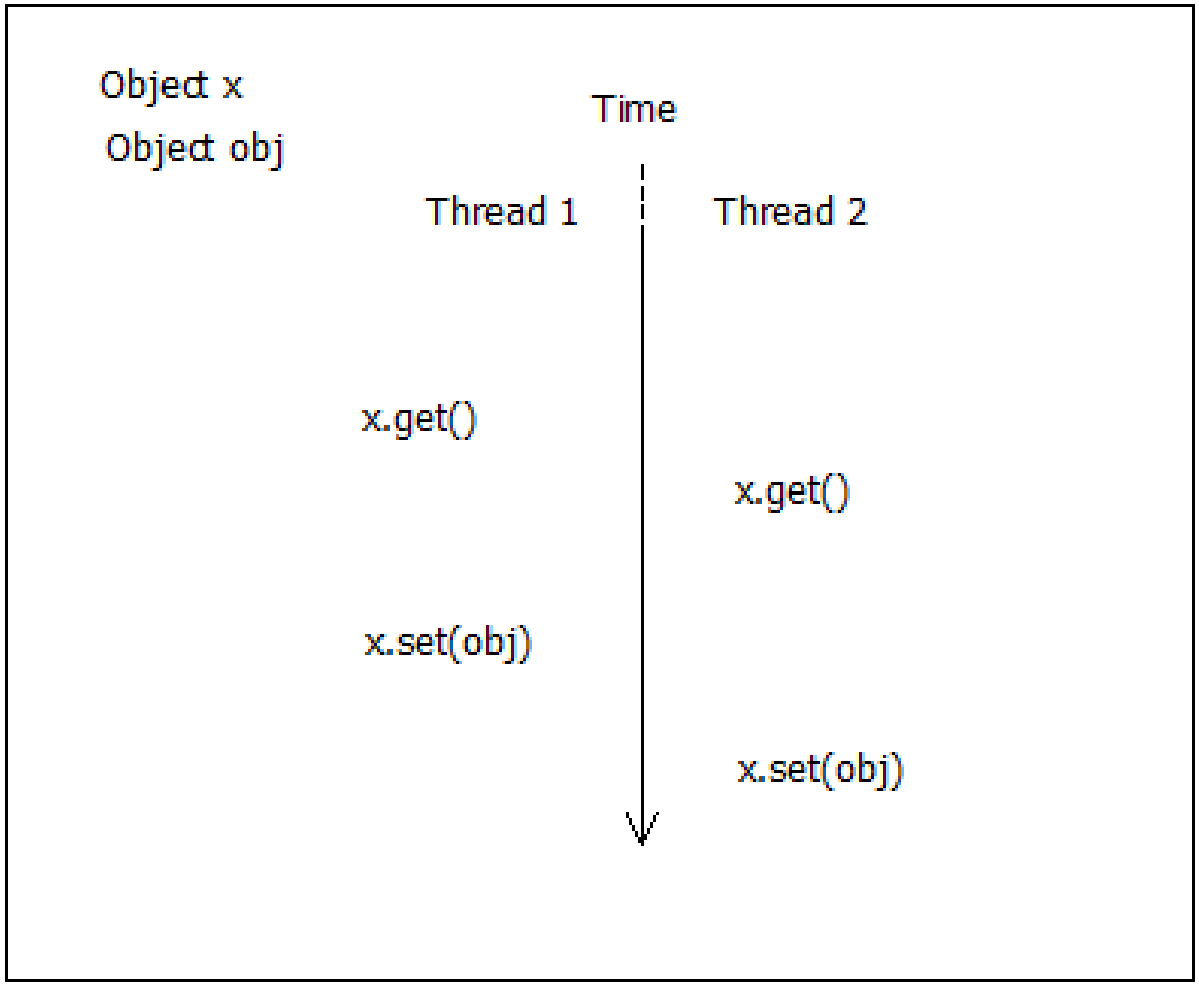
THREAD OBJECTS

```
public interface Runnable {
  void run();
}

public class Thread
        extends Object, implements Runnable {
  public Thread(Runnable target);
  public void start();
  public void run();
  public void interrupt();
  // other methods
}
```

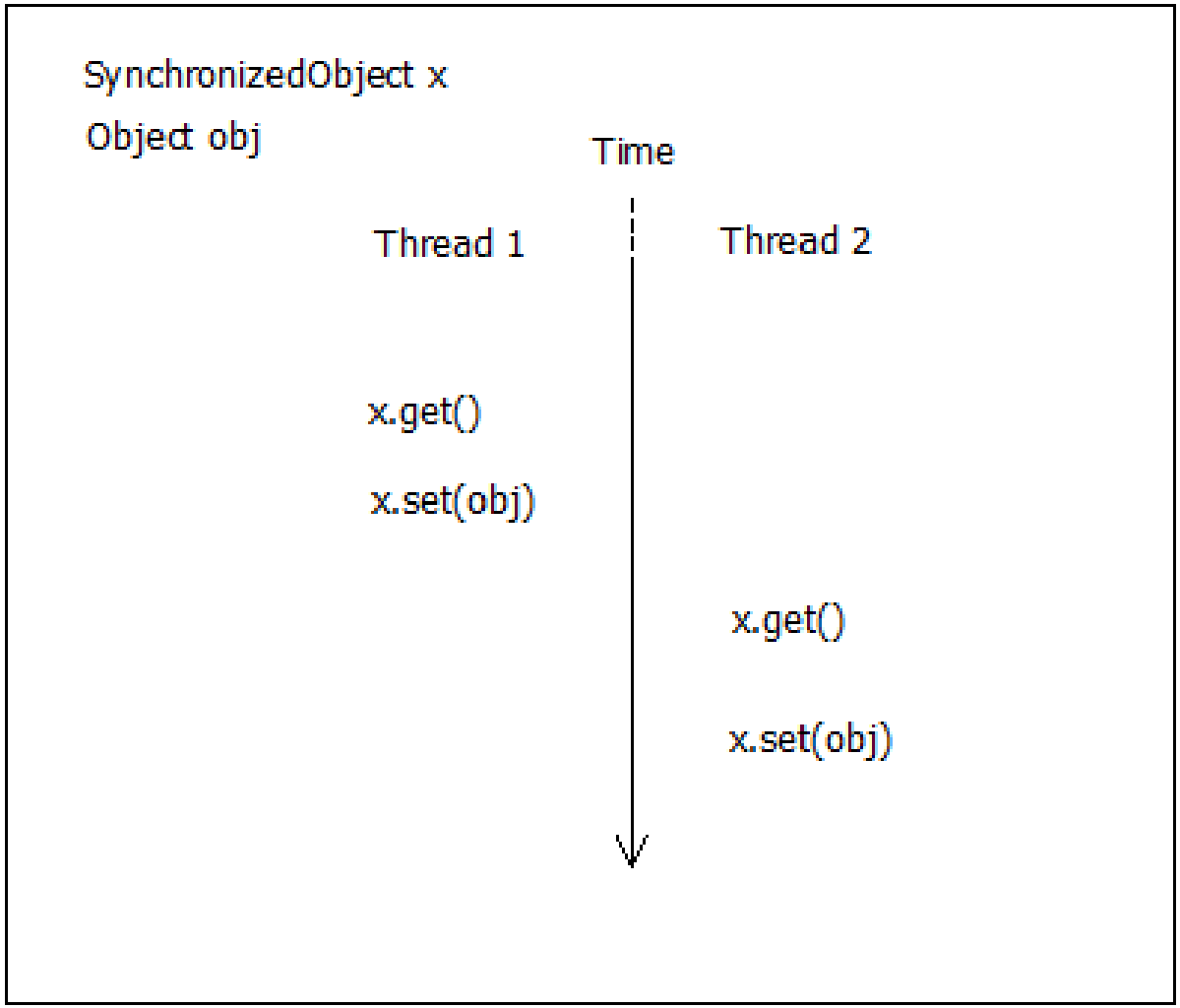
THREAD EXAMPLE

```
class TestRun implements Runnable {
  private long firstNumber;
  public TestRun(long firstNumber) {
  this.firstNumber = firstNumber;
  }
  public void run() {
  // compute suitable numbers larger than firstNumber
   . . .
  }
}

  TestRun p = new TestRun(147);
  new Thread(p).start();
```

UNSUNCHRONIZED OBJECTS



SYNCHRONIZED OBJECTS



SYNCHRONIZED OBJECT

```
public class SynchronizedObject {
  private Object state;
  public SynchronizedObject(Object initialState) {
        state=initialState;
  }
  public synchronized Object get() {
        return state; }
  public synchronized void set(Object obj) {
        state=obj ;}
  // methods inherited from Object:
  // public wait()
  // public void notifyAll()
  // other methods
}
```

SYNCHRONIZED CONTAINER

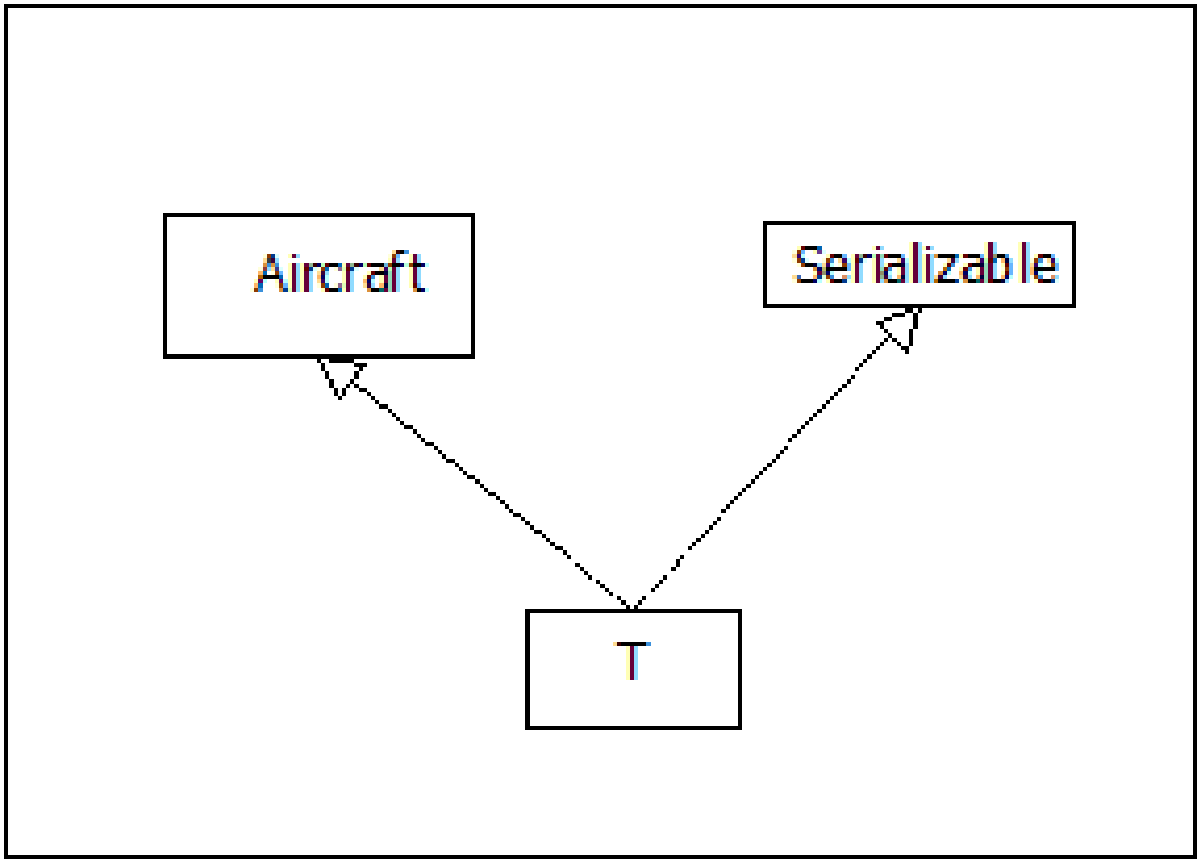
```
class SynchronizedContainer<T> {
  private Container<T> container = new Container<T>;
  public synchronized void add(T x) {
        container.add(x);
        notifyAll();
}
  public synchronized void remove(T x) {
        throws InterruptedException;
  { while (container.size() = 0)
        wait();
        container.remove(x);
  }
}
```

SYNCHRONIZATION and INHERITANCE

```
public class OrderedCollectionSync< T extends Comparable<T>>
                    extends OrderedCollection<T> {
  public OrderedCollectionSync() { super(); }
@Override
  public synchronized boolean contains(Object e) {
  if (e ≠ null) {return super.contains(e); }
  else return false;
}
@Override
public synchronized void add(T e) {
  super.add(e); }
@Override
public synchronized void remove(Object e) {
  super.remove(e); }
}
```

```
public class RunwayQueue<T extends Aircraft & Serializable >
        extends Queue<T> implements Serializable {
// fields, constructors and methods
  public synchronized void serialize()
{ try {
FileOutputStream fileStream = new FileOutputStream("RunwayData");
ObjectOutputStream objectStream = new ObjectOutputStream(fileStream);
objectStream.writeObject(this);
objectStream.close();
  }
  catch ( FileNotFoundException fileEx )
        {exception handling }
  catch ( IOException ioEx )
        { exception handling }
}
  public synchronized RunwayQueue<T> deserialize()
{// symmetric code
  }
}
```

MULTIPLE INHERITANCE



SYNCHRONIZED versus UNSYNCHRONIZED

```
public class TestRun implements Runnable {
  private OrderedCollection<Integer> collection;
  public TestRun(OrderedCollection<Integer> collection) {
        this.collection = collection;
  }
@Override
public void run() {
  //insertions to the list (0-100) and deletion of odd insertions
  synchronized(collection) {
        Integer toAdd;
        for (int i = 0; i < 100; i++) {
        toAdd = new Integer(i);
        collection.add(toAdd);
        if (i div 2 = 0)
        {collection.remove(new Integer(i -1));}
              }
        }
  }
}
```

THREAD CLASS

```
public class Thread
        extends Object, implements Runnable {
  public Thread(Runnable target);
  public void interrupt();
  public boolean isInterrupted();
  public void join()
        throws InterruptedException;
  public void run();
  public static void sleep()
        throws InterruptedException;
  public void start();
  public static void yield();
  // other methods
}
```

```
public static void main(String[] args) {
  private OrderedCollection<Integer> intCollection =
        new OrderedCollection<Integer>();
  Thread thread1 = new Thread(new TestRun(intCollection));
  Thread thread2 = new Thread(new TestRun(intCollection));
  thread1.start();
  thread2.start();
  try {//wait for threads to finish before checking the collection
        thread1.join();
        thread2.join();
  } catch (InterruptedException e) { exception handling }
  for (Integer i: intCollection)
        System.out.print(i);
  }
```

SYNCHRONIZED THREADS

```
OrderedCollectionSync<Integer> syncIntCollection =
        new OrderedCollectionSync<Integer>();
Thread thread1 = new Thread(new TestRun(syncIntCollection));
Thread thread2 = new Thread(new TestRun(syncIntCollection));
```

MESSAGES as OBJECTS

```
interface Message {
  Method m();
  Object receiver();
  Object[] arguments();
  int timeStamp();
}
```

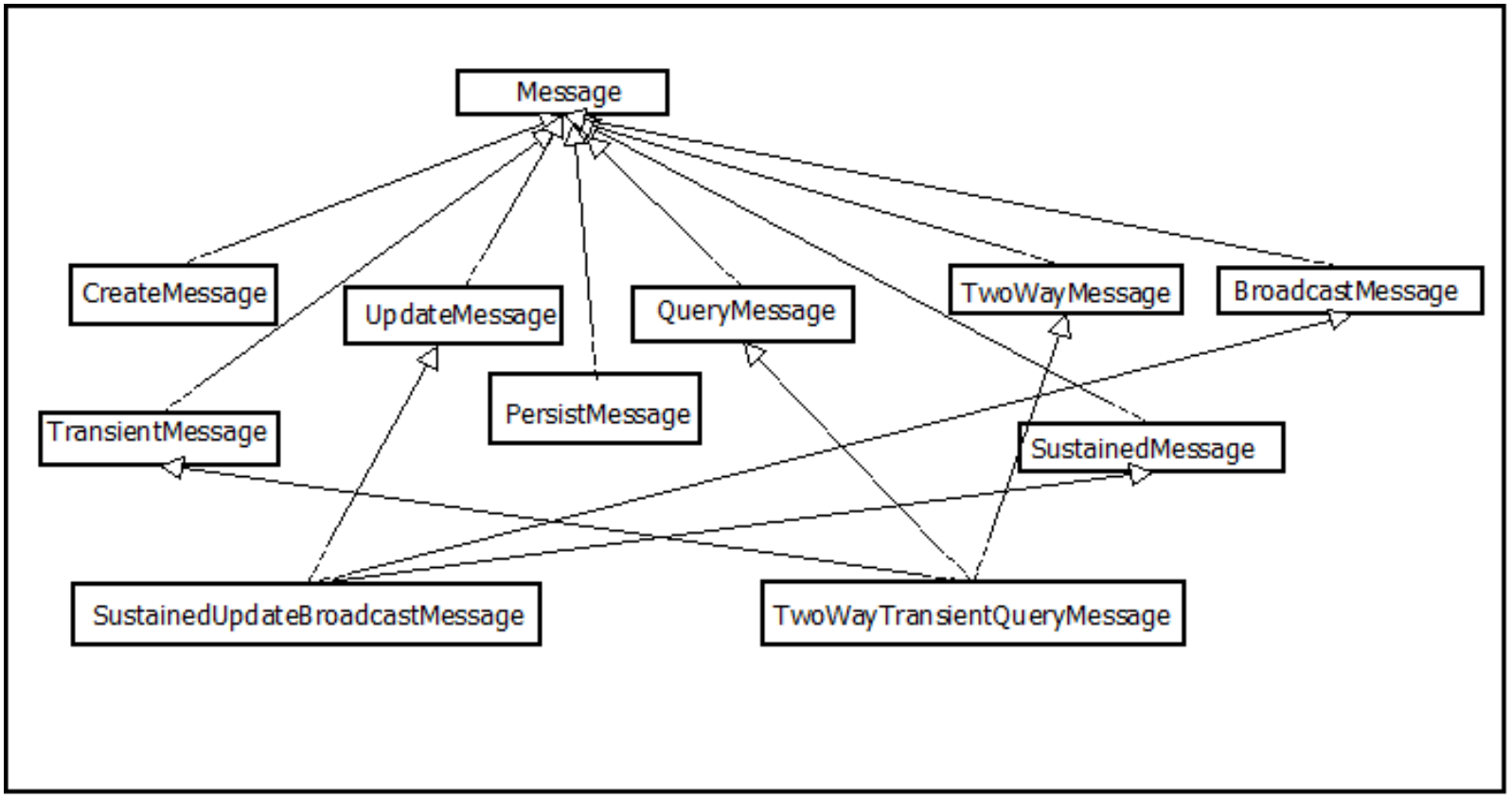
MESSAGES as OBJECTS

```
class MessageObject implements Message {
  MessageObject(Method m, Object receiver,
                Object[] arguments);
  int timeStamp();
  Method m();
  Object receiver();
  Object[] arguments();
}
```

CREATING MESSAGES

Message msg = **new** MessageObject(Method m, Object receiver, Object[] arguments)

TYPES of MESSAGES



MESSAGE TYPES

```
interface TwoWayMessage extends Message {
  boolean futureResolved();
  boolean setFuture(Object result);
  Object future() requires this.futureResolved();
}
```

MESSAGE TYPES

```
class TwoWayMessageObject implements TwoWayMessage {
  TwoWayMessageObject(Method m, Object receiver,
        Object[] arguments, int replyInterval);
  boolean futureResolved();
  boolean setFuture(Object result);
  Object future() requires this.futureResolved();
}
```

MESSAGE TYPES

```
class TransientMessageObject implements TransientMessage {
  TransientMessageObject(Method m, Object receiver,
                         Object[] arguments, int discoveryLifeTime);
  int discoveryLifeTime();
}
```

```
final class Class {
  String name();
  Method[] methods();
  Method getMethod(String name, Class[] arguments);
  . . .
  Assertion invariant();
}
```

```
final class Method {
  String name();
  Class declaringClass();
  Assertion preCondition();
  Assertion postCondition();
  Class[] arguments();
  Class result();
  Expression body();
  Object eval(Object receiver, Object[] args);
}
```

$$\mathcal{T} \vdash C : \mathit{Class},$$
$$\mathcal{T} \vdash \mathit{constructor}(D) : \mathit{Constructor},$$
$$\mathcal{T} \vdash \mathit{methods}(D) : \mathit{Method}[\,],$$
$$\mathcal{T} \vdash \mathit{methods}(D)\ \mathit{compatibleWith}\ \mathit{methods}(C),$$
$$\mathcal{T} \vdash \mathit{invariant}(D) : \mathit{Assertion}$$

$$\mathcal{T} \vdash \textbf{class}\ D\ \textbf{extends}\ C$$
$$\{\mathit{constructor}(D); \mathit{methods}(D); \mathit{invariant}(D)\} : \mathit{Class}$$

$$\frac{\begin{array}{c}\mathcal{T} \vdash C : \ Class, \\ \mathcal{T} \vdash A\ m(A1, A2, \ldots, An)\{\ldots\} \in methods(C), \\ \mathcal{T} \vdash \textbf{class}\ D\ \textbf{extends}\ C : \ Class, \\ \mathcal{T} \vdash B\ m(A1, A2, \ldots, An)\{\ldots\} \in methods(D)\end{array}}{\mathcal{T} \vdash B <: A}$$

$$\frac{\mathcal{T} \vdash \textbf{class } C < T \textbf{ extends } B > \quad \{constructor(C); methods(C); invariant(C)\} : Class < T <: B >, \quad \mathcal{T} \vdash D <: B < D/T >}{\mathcal{T} \vdash \textbf{class } C < D > \quad \{constructor(C) < D/T >; methods(C) < D/T >; invariant(C) < D/T >\} : Class}$$

$$\frac{\begin{array}{c}\mathcal{T} \vdash B : \ Class, \mathcal{T} \vdash Ai : \ Class \ for \ i = 1, 2, \ldots, n, \\ \mathcal{T} \cup \{x1 : A1, x2 : A2, \ldots, xn : An\} \vdash preCondition(m) : Assertion, \\ \mathcal{T} \cup \{x1 : A1, x2 : A2, \ldots, xn : An\} \vdash postCondition(m) : Assertion, \\ \mathcal{T} \cup \{x1 : A1, x2 : A2, \ldots, xn : An\} \vdash modifyList(m) : List < \\ modifyPair >, \\ \mathcal{T} \cup \{x1 : A1, x2 : A2, \ldots, xn : An\} \vdash expression(m) : B\end{array}}{\begin{array}{c}\mathcal{T} \vdash B \ m(A1 \ x1, A2 \ x2, \ldots, An \ xn) \\ \{preCondition(m); postCondition(m); modifyList(m); return \ expression(m)\} : \\ Method\end{array}}$$

TYPING MESSAGES

$$\frac{\begin{array}{c}\mathcal{T} \vdash x : A, \\ \mathcal{T} \vdash C\ m(D1, D2, \ldots, Dn)\{\ldots\} \in methods(A), \\ \mathcal{T} \vdash ei : Ei\ for\ i = 1, 2, \ldots, n, \\ \mathcal{T} \vdash Ei <: Di\ for\ i = 1, 2, \ldots, n\end{array}}{\mathcal{T} \vdash x.m(e1, e2, \ldots, en) :\ C}$$

ASYNCHRONOUS MESSAGES

$$\frac{\begin{array}{c}\mathcal{T} \vdash x : A, \\ \mathcal{T} \vdash C\ m(D1, D2, \ldots, Dn)\{\ldots\} \in methods(A), \\ \mathcal{T} \vdash ei : Ei\ for\ i = 1, 2, \ldots, n, \\ \mathcal{T} \vdash Ei <: Di\ for\ i = 1, 2, \ldots, n\end{array}}{\mathcal{T} \vdash x <= m(e1, e2, \ldots, en) :\ Message}$$

DIFFERENT MESSAGE TYPES

$$\frac{\begin{array}{c}\mathcal{T} \vdash x : A, \\ \mathcal{T} \vdash C\ m(D1, D2, \ldots, Dn)\{\ldots\} \in methods(A), \\ \mathcal{T} \vdash ei : Ei\ for\ i = 1, 2, \ldots, n, \\ \mathcal{T} \vdash Ei <: Di\ for\ i = 1, 2, \ldots, n\end{array}}{\mathcal{T} \vdash x <=>> m(e1, e2, \ldots, en) :\ BroadcastMessage}$$

BEHAVIORAL SUBTYPING: SUBCLASSES

$$\frac{\begin{array}{c}\mathcal{T} \vdash \textbf{class } D \textbf{ extends } C : \ Class, \\ \mathcal{T} \vdash boundVariables(invariant(D)) = \{x_1, x_2, \ldots, x_n\}, \\ \mathcal{T} \vdash boundVariables(invariant(C)) \subset boundVariables(invariant(D)), \\ \mathcal{T} \vdash x_i : A_i \text{ for } i = 1, 2, \ldots, n, \\ \mathcal{T} \vdash a_i : A_i \text{ for } i = 1, 2, \ldots, n\end{array}}{invariant(D)[a_i/x_i] \ \Rightarrow \ invariant(C)[a_i/x_i]}$$

BEHAVIORAL SUBTYPING

$$\frac{\begin{array}{c}\mathcal{T} \vdash \textbf{class } D \textbf{ extends } C : \ Class, \\ \mathcal{T} \vdash A\ m(A1, A2, \ldots, An)\{\ldots\} \in methods(C), \\ \mathcal{T} \vdash freeVariables(precondition_D(m)) = \{x_1, x_2, \ldots, x_n\}, \\ \mathcal{T} \vdash freeVariables(precondition_C(m)) \subseteq \\ freeVariables(precondition_D(m)), \\ \mathcal{T} \vdash x_i : A_i \text{ for } i = 1, 2, \ldots, n, \\ \mathcal{T} \vdash a_i : A_i \text{ for } i = 1, 2, \ldots, n\end{array}}{preCondition_D(m)[a_i/x_i] \ \Leftrightarrow \ preCondition_C(m)[a_i/x_i]}$$

$$\mathcal{T} \vdash \textbf{class } D \textbf{ extends } C : \ Class,$$
$$\mathcal{T} \vdash A\ m(A1, A2, \ldots, An)\{\ldots\} \in methods(C),$$
$$\mathcal{T} \vdash freeVariables(postcondition_D(m)) = \{x_1, x_2, \ldots, x_n\},$$
$$\mathcal{T} \vdash freeVariables(postcondition_C(m)) \subset freeVariables(postcondition_D(m)),$$
$$\mathcal{T} \vdash x_i : A_i \text{ for } i = 1, 2, \ldots, n,$$
$$\mathcal{T} \vdash a_i : A_i \text{ for } i = 1, 2, \ldots n$$

$$postCondition_D(m)[a_i/x_i] \ \Rightarrow \ postCondition_C(m)[a_i/x_i]$$

```
abstract class Ambient<T extends ServiceObject> {
  abstract boolean filter(T x);
  Set<Message> messages();
  Set<T> communicationRange();
  Set<T> reach();
  invariant (∀ T x)
  ((x ∈ this.reach()) ⇔
  (this.filter(x) ∧ (x ∈ this.communicationRange())))
}
```

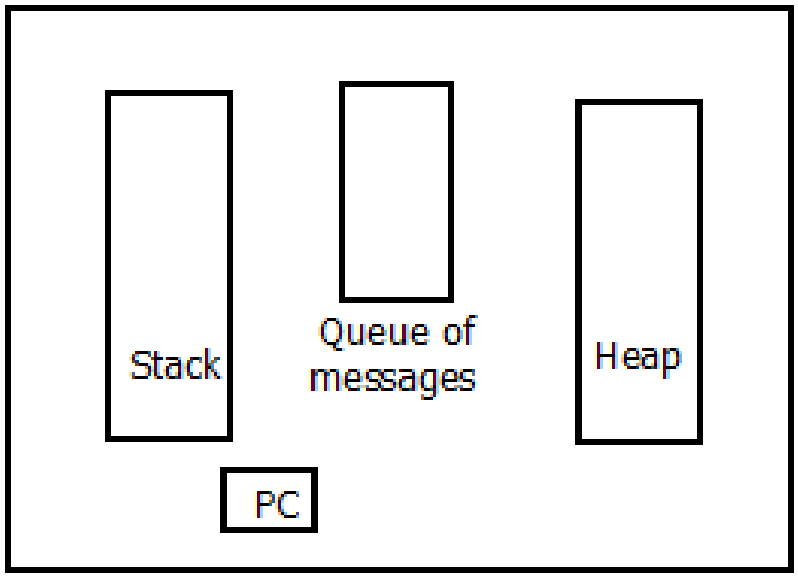
SAMPLE AMBIENT

```
class StockBroker extends ServiceObject {
  int quote(String stock);
  int responseTime();
  . . .
}

class StockBrokerAmbient extends Ambient<StockBroker> {
  String[] displayStocks(){. . .};
  requestQuote(String stock){. . .};
  boolean filter(StockBroker x)
  {return x.responseTime() ≤ 10; }
}
```

```
StockBrokerAmbient stockbrokers =
        new StockBrokerAmbient();


Method requestQuote = getClass("StockBrokerAmbient").getMethod(
        "requestQuote", getClass("String"));
Message requestQuoteMsg = new MessageObject(requestQuote,
        stockBrokers,stock)
```

```
class ConcurrentObject {
  private VirtualMachine VM();
}
```

CONCURRENT OBJECTS



```
final class VirtualMachine {
  int currentTime();

  private Stack<StackValue< T >> VMstack();
  private Heap<Object> VMheap();
  private Queue<Message> messages();

  enum Exception = (noException, preCondition, postCondition,
        invariant, expiredMessage);

  private boolean executeMessage(Message msg);
  private boolean setException(Exception e);
  Exception getException();
  . . .
}
```

```
class ServiceObject extends ConcurrentObject {. . . }

class Ambient <T extends ServiceObject>
        extends ConcurrentObject {
  . . . }
```

MOBILE OBJECTS

```
class MobileObject extends ConcurrentObject {
  Location loc ();
}
```

```
class Region <T> extends Ambient<T> {
  Set<ConcurrentObject> objects();
  boolean withinRegion(MobileObject x);
  invariant
  (∀ MobileObject x)(this.withinRegion(x) ⇒
        (x ∈ this.objects()))
}
```