```
public class CameraAgent extends Agent implements Camera {

    private Map<Nest, Integer> nests = new HashMap<Nest, Integer>();
    private Map<Kit, Integer> kits = new HashMap<Kit, Integer>();
    private NestAgent nestAgent;
    private KitRobotAgent kitRobotAgent;

    // ********** MESSAGES *********

    public void msgNestIsFull(Nest nest, int nestNum) {
        nests.put(nest, nestNum);
        stateChanged();
    }

    public void msgKitIsFull(Kit kit, int kitNum) {
        kits.put(kit, kitNum);
        stateChanged();
    }

    // ********* SCHEDULER *********
    protected boolean pickAndExecuteAnAction() {
        for (Map.Entry<Kit, Integer> entry : kits.entrySet()) {
            inspectKit(entry.getKey(), entry.getValue());
            return true;
        }
        for (Map.Entry<Nest, Integer> entry : nests.entrySet()) {
            inspectNest(entry.getKey(), entry.getValue());
        }

        return false;
    }

    // ********** ACTIONS **********

    private void inspectKit(Kit kit, int kitNum) {
//        DoInspectKit(kitNum);
        //check if all the correct parts
        kitRobotAgent.msgKitInspected(kit, true);
        stateChanged();
    }

    private void inspectNest(Nest nest, int nestNum) {
//        DoInspectNest(nestNum);
        //check if all the correct parts
        nestAgent.msgNestInspected(true);
        stateChanged();
    }

    // ************ MISC ***********

    public void setNestAgent(NestAgent agent) {
        nestAgent = agent;
    }

    public void setKitRobotAgent(KitRobotAgent agent) {
        kitRobotAgent = agent;
    }
}



=========================================================================================
=========================================================================================
=========================================================================================




public class ConveyorAgent extends Agent implements Conveyor {

    private enum Event {

        emptyKit, verifiedKit
    };
    private List<Event> eventQueue = new ArrayList<Event>();
    private List<Kit> kits = new ArrayList<Kit>();
    private Kit tempKit;
    private KitRobotAgent kitRobotAgent;

    // ********** MESSAGES *********


    public void msgNeedEmptyKit() {
        eventQueue.add(Event.emptyKit);
        stateChanged();
    }

    public void msgHereIsVerifiedKit(Kit kit) {
        eventQueue.add(Event.verifiedKit);
        tempKit = kit;
        stateChanged();
    }

    // ********* SCHEDULER *********

    protected boolean pickAndExecuteAnAction() {
        for (Event e : eventQueue) {
            if (e == Event.emptyKit) {
                giveEmptyKit();
                return true;
            } else if (e == Event.verifiedKit) {
                acceptVerifiedKit();
                return true;
            }
        }

        return false;
    }

    // ********** ACTIONS **********

    private void giveEmptyKit() {
//        DoGiveEmptyKit();
//        Location loc = kits.get(0).getLocation();
//        kitRobotAgent.msgHereIsKit(loc);
        stateChanged();
    }


    private void acceptVerifiedKit() {
//        DoAcceptVerifiedKit();
        kits.add(tempKit);
        stateChanged();
    }

    // ************ MISC ***********

    public void setKitRobotAgent(KitRobotAgent agent) {
        kitRobotAgent = agent;
    }
}



===========================================================================================
===========================================================================================
===========================================================================================



public class KitRobotAgent extends Agent implements KitRobot {

    List<myKit> kits = new ArrayList<myKit>();

    enum KitStatus {

        empty, complete, verified, error
    };

    private class myKit {

        Kit kit;
        KitStatus status;
        int conveyorLoc;
        int kittingStandNum;

        public myKit(Kit kit, int conveyorLoc) {
            this.kit = kit;
            this.status = KitStatus.empty;
            this.conveyorLoc = conveyorLoc;
            this.kittingStandNum = -1;
        }
    }
    private boolean needEmptyKit = false;
    private ConveyorAgent conveyor;
    private CameraAgent camera;
    private PartsAgent partsAgent;

    // ********** MESSAGES *********
    public void msgNeedEmptyKit() {
        needEmptyKit = true;
        stateChanged();
    }

    public void msgHereIsEmptyKit(Kit kit, int loc) {
        kits.add(new myKit(kit, loc));
        stateChanged();
    }

    public void msgKitIsComplete(Kit kit) {
        for (myKit k : kits) {
            if (kit == k.kit) {
                k.status = KitStatus.complete;
                break;
            }
        }
        stateChanged();
    }

    public void msgKitInspected(Kit kit, boolean result) {
        for (myKit k : kits) {
            if (kit == k.kit) {
                if (result) {
                    k.status = KitStatus.verified;
                    break;
                } else {
                    k.status = KitStatus.error;
                    break;
                }
            }
        }
        stateChanged();
    }

    // ********* SCHEDULER *********

    protected boolean pickAndExecuteAnAction() {
        if (!kits.isEmpty()) {
            for (myKit k : kits) {
                if (k.status == KitStatus.verified) {
                    removeVerifiedKit(k);
                    return true;
                }
            }
            for (myKit k : kits) {
                if (k.status == KitStatus.complete) {
                    moveFullKitToInspection(k);
                    return true;
                }
            }
            for (myKit k : kits) {
                if (k.status == KitStatus.empty) {
                    giveEmptyKit(k);
                    return true;
                }
            }
        } else if (needEmptyKit) {
            getEmptyKit();
            return true;
        }

        return false;

    }

    // ********** ACTIONS **********
    private void removeVerifiedKit(myKit k) {
//        DoRemoveVerifiedKit();
        conveyor.msgHereIsVerifiedKit(k.kit);
        stateChanged();
    }

    private void moveFullKitToInspection(myKit k) {
//        DoMoveFullKitToInspection();
        camera.msgKitIsFull(k.kit, k.kittingStandNum);
        stateChanged();
    }

    private void giveEmptyKit(myKit k) {
//        DoGiveEmptyKit();
        partsAgent.msgEmptyKitReady(k.kittingStandNum);
        stateChanged();
    }

    private void getEmptyKit() {
```

```
//        DoGetEmptyKit();
        conveyor.msgNeedEmptyKit();
        stateChanged();
    }
    // ************ MISC ***********
}
```

```
==========================================================================================
==========================================================================================
==========================================================================================
```

```
FEEDER AGENT:
DATA:
List<myParts> parts
Lane leftLane;
Lane rightLane;
Gantry gantry;
enum SendTo{leftLane,rightLane,none};

class myParts{
        Part part_type;
        int quantity;
        int supplyAmount;
        boolean send;
        SendTo sendTo;

}

MESSAGES:
1.      public void msgNeedPart(Part part,Lane lane)

/*
 * Search in the myParts list and see if the request can be fulfilled
 */
        if ∃ p in myParts  → p.part_type.type=part.type

if(lane==this.leftLane)
                         then
                                 p.send=true;
                                 p.sendTo=SendTo.leftLane;
                                 p.supplyAmount=leftLane.capacity;
                                 return;


                         //is the message from the right lane?
                         if(lane=this.rightLane)
                         then
p.send=true;
                                 p.sendTo=SendTo.rightLane;
                                 p.supplyAmount=rightLane.capacity;
                                 return;

2.      public void msgHereAreParts(Part part, int quantity)

        //add to the existing list of parts if the parts already exist

if ∃ p in myParts → p.part_type.type=part.type
        then
                p.quantity=p.quantity+quantity;
return;

//create a new type if the current list does not contain parts of this type.
parts.add(new myParts(part,quantity));


SCHEDULER:

if ∃ p in myParts → p.send=true
if(p.sendTo=SendTo.leftLane)
                if(p.quantity<leftLane.capacity)
                then
needPart(p);
                else
                        sendPartToLeftLane(p);

        if(p.sendTo=SendTo.rightLane)
                if(p.quantity<rightLane.capacity)
                then
```

```
needPart(p);
                else
                        sendPartToRightLane(p);

ACTIONS:

1.      needPart(myParts p)

        gantry.msgNeedPart(p.part_type);

2.      sendPartToLeftLane(myParts p)

doSendPartsToLeftLane();// ANIMATION
leftLane.msgHereAreParts(p.part_type,p.quantity);
        p.send=false;
        p.sendTo=SendTo.none;
        //update the myParts object
        p.quantity=p.quantity-p.supplyAmount;

3.      sendPartToRightLane(myParts p)

doSendPartsToRightLane();// ANIMATION
rightLane.msgHereAreParts(p.part_type,p.quantity);
        p.send=false;
        p.sendTo=SendTo.none;
        //update the myParts object
        p.quantity=p.quantity-p.supplyAmount;



=========================================================================================
=========================================================================================
=========================================================================================




GANTRY AGENT:
DATA:
List<myParts> parts
Feeder feeder

/* to hold the info about its list of parts*/
class myParts{
                Part part_type;
                int quantity;
                int supplyAmount;
                boolean send;
}

MESSAGES:

1.      public void msgNeedPart(Part part)

if ∃ p in myParts → p.part_type.type=part.type
        then
p.send=true;
                p.supplyAmount=feeder.capacity;
                return;

SCHEDULER:

if ∃ p in myParts → p.send=true
        then
                supplyPart(p);

ACTIONS:

1.      supplyPart(myParts p)
                DoSendPartsToFeeder(); // ANIMATION
feeder.msgHereAreParts(p.part_type,p.supplyAmount);
                p.send=false;
                //update the myParts object
        p.quantity=p.quantity-p.supplyAmount;



=========================================================================================
=========================================================================================
=========================================================================================
LANE AGENT:
DATA:
List<myParts> parts
Nest nest;

class myParts{
                Part part_type;
                int quantity;
                int supplyAmount;
                boolean send;


        }
MESSAGES:
1.      msgNeedPart(Part part)
if ∃ p in myParts → p.part_type.type=part.type
        then
p.send=true;
                p.supplyAmount=nest.capacity;

2.      msgHereAreParts(Part part, int quantity)
        //add to the existing list of parts if the parts already exist

if ∃ p in myParts → p.part_type.type=part.type
        then
                p.quantity=p.quantity+quantity;
return;

//create a new type if the current list does not contain parts of this type.
parts.add(new myParts(part,quantity));

SCHEDULER:
1.      if ∃ p in myParts → p.send=true
        then
                supplyPart(p);

ACTIONS:
1.      supplyPart(myParts part)
                doSendPartsToNest();// ANIMATION
                nest.msgHereAreParts(part.part_type, part.supplyAmount);
                part.send=false;
                //update the myParts object
                part.quantity=part.quantity-part.supplyAmount;
```

```
==========================================================================================
==========================================================================================
==========================================================================================
```

```
public class Part {
public enum Type {p1, p2, p3, p4, p5, p6, p7, p8};
public Type type;
public boolean inKit;
public int size;
//public boolean good;

public Part(Type t, boolean inkit, int size){
        this.type = t;
        this.inKit = false;
        this.size = size;
}

}
```

```
==========================================================================================
==========================================================================================
==========================================================================================
```

KIT CLASS

```
public class Kit {
```

```
public Part parts[];
public enum Status {empty, full, inspected};
public Status status;
public int kitNeedsParts;
public static enum KittingStandNumber {none, one, two, three};
public KittingStandNumber kittingStandNumber;

public Kit(Part p[]){
this.kittingStandNumber = KittingStandNumber.none;
this.status = Status.empty;
this.kitNeedsParts = p.length;
for (int i=0; i<p.length; i++){
        parts[i] = p[i];
}
}
```

```
==========================================================================================
==========================================================================================
==========================================================================================
```

NESTAGENT CLASS

```
public class NestAgent {
        PartsAgent partsagent;
        LaneAgent lane;
        CameraAgent camera;
        private int threshold;
        private Part part;
        private int howMany = 0;
        private int nestNumber;
        private boolean verified = false;


    private List<Part> parts =
            Collections.synchronizedList(new ArrayList<Part>());
    PartsAgent partsagent;
    LaneAgent lane;
    CameraAgent camera;
    private int threshold;
    private Part part;
    private int howMany = 0;
    private int nestNumber;
    private List<Nest> nests = Collections.synchronizedList(new ArrayList<Nest>());

    NestAgent(int nestNum, LaneAgent lane) {
        this.nestNumber = nestNum;
        this.lane = lane;
    }
//messages

        public void msgNeedPart(Part p){
                if (part == p)
                        partsagent.msgHereAreParts(part, howMany);
                else
                        lane.msgNeedPart(p);
        }

        public void msgHereAreParts(Part p, int quantity){
                part = p;
                threshold = 10/p.size;
                howMany += quantity;
                if (howMany>threshold){
                        this.lane.msgRejectParts(howMany - threshold);
                        howMany = threshold;
                        camera.msgNestIsFull(nestNumber);
                }

        }

        public void msgNestVerified(boolean result){
                if (result)
                partsagent.msgHereAreParts(part, howMany);
                verified = result;
        }
        public void msgPurgeNest(){
                part = null;
                howMany = 0;

        }
        //scheduler
        protected boolean pickAndExecuteAnAction() {

                return false;
        }

        //actions

        public void setPartsAgent(PartsAgent parts){
                this.partsagent = parts;
        }

        public void setPart(Part p){
                part = p;
        }
```

```
===========================================================================================
===========================================================================================
===========================================================================================
```

PartsAgent Class

```
public class PartsAgent extends Agent {

    KitRobotAgent kitagent;
    Kit kit;
    NestAgent nest;

private List<NestAgent> myNests = Collections.synchronizedList(new ArrayList<NestAgent>(8));
private List<configFile> configInfo =
Collections.synchronizedList(new ArrayList<configFile>());
private Map<Part, Integer> inventory = new HashMap<Part, Integer>();
Part grips[];

    private List<ConfigFile> configInfo =
            Collections.synchronizedList(new ArrayList<ConfigFile>());
    private Map<Part, Integer> inventory = new HashMap<Part, Integer>();
    Part grips[];


//Messages
public void msgHereIsConfig(configFile){
        configInfo.add(configFile);
        stateChanged();
}

public void msgHereAreParts(Part p, int quantity){
        inventory.put(Part, quantity);
        stateChanged();
}

public void msgEmptyKitReady(int num) {
        switch (num) {
            case 1:
                kit.kittingStandNumber = Kit.KittingStandNumber.one;
                break;
            case 2:
                kit.kittingStandNumber = Kit.KittingStandNumber.two;
                break;
            case 3:
                kit.kittingStandNumber = Kit.KittingStandNumber.three;
                break;
            default:
                kit.kittingStandNumber = Kit.KittingStandNumber.none;
        }
        stateChanged();
    }

public void msgHereIsNewKit(Kit k){
kit = k;
stateChanged();
}
//Scheduler
```

```
protected boolean pickAndExecuteAnAction() {

        if (!configInfo.isEmpty()){
                setConfiguration();
                return true;
        }


        if (!inventory.isEmpty() && kit.status == Kit.Status.empty && kit.kittingStandNumber!=Kit.KittingStandNumber.none)
{
                int n = 4; //4 grips
                int grip = 0;  // grip index
                if (kit.kitNeedsParts<4) //if
                        n = kit.kitNeedsParts;
                for (int i=0; i<kit.parts.length; i++){
                        if (!kit.parts[i].inKit)
                                if (inventory.containsKey(kit.parts[i])){
                                pickUpPart(kit.parts[i], grip);
                                kit.parts[i].inKit = true;
                                grip++;
                                if (grip == n)
                                        putPartsInKit(n);
                                        return true;
                                }
                }
        }

        if (kit.status == Kit.Status.full){
                giveKitToKitAgent();

        }

        return false;
}
//Actions

private void setConfiguration(){
        if (configInfo.hasNewKit()){
                {kit = configInfo.getKit();
                kitagent.msgNeedEmptyKit();}
        for (int i=0; i<kit.parts.length; i++){
                myNests(i).msgNeedPart(kit.parts[i]);
                myNests(i).setPart(kit.parts[i]);

        }}
}


private void giveKitToKitAgent(){
        kitagent.msgKitIsComplete();
}

private void pickUpPart(Part p, int g){
        grips[g] = p;
        doPickUpPart(p);
        if (kit.kitNeedsParts == 0){
                kit.status = Kit.Status.full;
        }
        inventory.put(p, inventory.get(p)-1);
}

private void putPartsInKit(int n){
        for (int i =0; i<n; i++){
        doPutPartInKit(grips[i]);
        kit.kitNeedsParts--;}
}
```