```java
package framework.tracking;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.util.concurrent.ConcurrentHashMap;

public class Communication {
    private final String trackingIpAddress_ = "230.230.230.230";
    private final int port_ = 5230;
    private final int maxTargets_ = 8;
    private MulticastSocket socket_;
    private boolean listen_ = true;

    private ConcurrentHashMap<Integer, String> values_;

    /**
     * Constructs a network instance and starts listening to the tracking system
     * message on a different thread.
     *
     * The constructor will start the thread itself.
     *
     * @param adapterIpAddress
     *            the ip v4 address of this machine of the adapter which is
     *            connected to the tracking system.
     * @throws IOException
     */
    public Communication(String adapterIpAddress) throws IOException {
        // This call makes sure IPv4 is used.
        System.setProperty("java.net.preferIPv4Stack", "true");

        socket_ = new MulticastSocket(port_);
        socket_.setInterface(InetAddress.getByName(adapterIpAddress));
        socket_.setReuseAddress(true);
        socket_.setSoTimeout(15000);
        socket_.joinGroup(InetAddress.getByName(trackingIpAddress_));
        values_ = new ConcurrentHashMap<>();

        new Thread(() -> {
            listen();
        }).start();
    }

    private void listen() {
        byte[] buffer = new byte[maxTargets_ * 256];

        while (listen_) {
            DatagramPacket msgPacket = new DatagramPacket(buffer, buffer.length);
            try {
                socket_.receive(msgPacket);
            } catch (IOException e) {
                e.printStackTrace();
            }

            String msg = new String(buffer, 0, buffer.length);
            msg = msg.replaceAll("\r|\n", "");

            if (msg.matches(".*6d\\s[1-9].*")) {
                String values = msg.replaceFirst("[^\\[]*\\[", "").replaceAll("\\s*3d.*", "");
                String frame = msg.replaceFirst("fr\\s", "").replaceAll("ts.*", "");

                String s[] = values.split(" \\[");
                for (String part : s) {
                    String id = part.replaceAll("\\s.*", "");

                    if (id.matches("\\d*")) {
                        values_.put(new Integer(id), frame + ";" + part);
                    } else {
                        System.err.println("not an ID: " + id);
                    }
                }
            }
```

```
        }
    }

    public void shutdown() {
        listen_ = false;
        if (socket_ != null && !socket_.isClosed()) {
            socket_.close();
        }
    }

    public String getValue(int trackableObjectID) {
        return values_.get(new Integer(trackableObjectID));
    }
}
```

```java
package framework.tracking;

import java.util.ArrayList;

import engine.gameObject.trackable.TrackableGameObject;
import framework.Position;

public class DataInterpreter implements Runnable{
    private final double secondsPerFrame = 1.0 / 60.0;
    private final int resolution_ = 50000;
    private final int edgeResolution_ = -1000;

    private final int trackableObjectID_;
    private final Communication communication_;

    private String prevData_ = "";
    private TrackingData prevTrackingData_ = new TrackingData();
    private int prevFrame_ = 0;
    private double lostDataSpeedFactor_ = 0.95;

    private ArrayList<TrackableGameObject> observer_ = new ArrayList<>();

    /**
     * Constructs an DataInterpreter instance and adds the passed observers to this instance.
     *
     * @param communication the communication to poll data from.
     * @param trackableObjectID the tracking ID of the trackable object;
     */
    public DataInterpreter(Communication communication, int trackableObjectID, TrackableGameObject...
    observers) {
        communication_ = communication;
        trackableObjectID_ = trackableObjectID;

        for (TrackableGameObject observer : observers) {
            if (!observer_.contains(observer)) {
                observer_.add(observer);
            }
        }
    }

    @Override
    /**
     * Polls for new data from the network class, parses it and distributes it
     * to its observers on a different Thread.
     **/
    public void run() {
        String data;
        while (true) {
            data = communication_.getValue(trackableObjectID_);
            if (data != null && !data.equals(prevData_)) {
                prevData_ = data;
                TrackingData trackingData = new TrackingData();
                Position pos = new Position();

                int frame = Integer.parseInt(data.replaceAll(";.*", "")) % resolution_;
                int frameDiff = frame - prevFrame_;

                if (frameDiff < edgeResolution_) {
                    frameDiff += resolution_;
                }

                if (frameDiff > 0) {
                    prevFrame_ = frame;
                    String coordinates = data.replaceFirst("[^(\\[)]*(\\[)", "").replaceAll("(\\]).*",
                    "");
                    String matrix = data.replaceFirst("([^(\\[)]*(\\[)){2}", "").replaceAll("\\].*", "");

                    String[] valueParts = coordinates.split("\\s");
                    String[] matrixValues = matrix.split("\\s");

                    if (valueParts.length == 6) {
                        pos.setX((int) Double.parseDouble(valueParts[0]));
                        pos.setY((int) Double.parseDouble(valueParts[1]));
```

```
                        pos.setZ((int) Double.parseDouble(valueParts[2]));
                        trackingData.setPosition(pos);

                        double[][] rotationMatrix = {
                                { Double.parseDouble(matrixValues[0]),
                                Double.parseDouble(matrixValues[3]),
                                        Double.parseDouble(matrixValues[6]) },
                                { Double.parseDouble(matrixValues[1]),
                                Double.parseDouble(matrixValues[4]),
                                        Double.parseDouble(matrixValues[7]) },
                                { Double.parseDouble(matrixValues[2]),
                                Double.parseDouble(matrixValues[5]),
                                        Double.parseDouble(matrixValues[8]) } };

                        trackingData.setRotationMatrix(rotationMatrix);

                        if (prevTrackingData_.isSignificantlyDifferent(trackingData, 1)) {
                            double secondsDiff = frameDiff * secondsPerFrame;
                            // distance [mm]
                            double distance =
                            prevTrackingData_.getPosition().getDistance(trackingData.getPosition());
                            // speed [m/s]
                            trackingData.setSpeed(distance / 1000.0 / secondsDiff);
                            prevTrackingData_ = trackingData;
                            notifyObservers(prevTrackingData_);
                        }
                    }
                }
            } else {
                prevTrackingData_.setSpeed(prevTrackingData_.getSpeed() * lostDataSpeedFactor_);
                notifyObservers(prevTrackingData_);
            }
        }
    }

    private void notifyObservers(TrackingData point) {
        for (TrackableGameObject observer : observer_) {
            observer.update(point);
        }
    }
}
```

```java
package framework.tracking;

import framework.Position;

public class TrackingData
{
    private final int   matrixDimension_    = 3;

    private double[][]  rotationMatrix_;
    private Position    position_;
    private double      speed_;

    /**
     * Constructs a TrackingData instance.
     *
     * </br>
     * <b>Tracking coordinate system.</b>
     */
    public TrackingData()
    {
        rotationMatrix_ = new double[matrixDimension_][matrixDimension_];
        position_ = new Position();
    }

    /**
     * Constructs a TrackingData instance. </br>
     * <b>Tracking coordinate system.</b>
     *
     * @param pos
     * @param rotationMatrix
     *            a 3x3 rotation matrix.
     */
    public TrackingData(Position pos, double[][] rotationMatrix)
    {
        setRotationMatrix(rotationMatrix);
        position_ = pos;
    }

    /**
     * Returns true if the difference to the x, y or z axis of the otherPoint is
     * greater than the passed difference. False otherwise.
     *
     * @param otherData
     *            the TrackingData to compare with.
     * @param difference
     *            the difference to compare with.
     * @return
     */
    public boolean isSignificantlyDifferent(TrackingData otherData, int difference)
    {
        return (Math.abs(position_.getX() - otherData.getPosition().getX()) > difference)
                || (Math.abs(position_.getY() - otherData.getPosition().getY()) > difference)
                || (Math.abs(position_.getZ() - otherData.getPosition().getZ()) > difference);
    }

    /**
     * Sets the rotation matrix.
     *
     * @param rotationMatrix
     *            a 3x3 rotation matrix.
     */
    public void setRotationMatrix(double[][] rotationMatrix)
    {
        if (rotationMatrix.length != 3 || rotationMatrix[0].length != 3)
        {
            throw new IllegalArgumentException("matrix dimaensions must be " + matrixDimension_ + ".");
        }
        rotationMatrix_ = rotationMatrix;
    }

    public TrackingData clone()
    {
        TrackingData clone = new TrackingData(position_, rotationMatrix_);
```

```
        clone.setSpeed(speed_);
        return clone;
    }

    /**
     * @return the current position
     */
    public Position getPosition()
    {
        return position_;
    }

    /**
     * Sets the current position
     *
     * @param position
     */
    public void setPosition(Position position)
    {
        position_ = position;
    }

    /**
     * Sets the speed.
     *
     * @param speed
     *            in m/s.
     */
    public void setSpeed(double speed)
    {
        speed_ = speed;
    }

    /**
     * @return the current speed in <b>m/s</b>.
     */
    public double getSpeed()
    {
        return speed_;
    }

    /**
     * @return the rotationMatrix
     */
    public double[][] getRotationMatrix()
    {
        return rotationMatrix_;
    }
}
```