```
/**
 * Update temperature readings for an asset
 * @param {org.reliance.network.TempReading} tempread
 * @transaction
 */

async function TempReading(tempread){

  // Declare global variables and get factory class
  const factory = getFactory();
  const NS = 'org.reliance.network';

  //Fetching contract and shipment for which the temperature data is to be captured
  let contractRegistry = await getAssetRegistry('org.reliance.network.Contract');
  let shipmentRegistry = await getAssetRegistry('org.reliance.network.Shipment');
  let fetchContract = await contractRegistry.get('tempread.shipment.contract.contractID');
  let fetchShipment = await shipmentRegistry.get('tempread.shipment.shipmentID');

  //Creating new transaction to capture the data and push into the array in Shipment asset data
structure
  let transaction =factory.newTransaction(NS, 'TempReading');
  transaction.celcius = tempread.celcius;
  transaction.latitude = tempread.latitude;
  transaction.longitude = tempread.longitude;
  transaction.readingtime = tempread.readingtime;


  //Push the temperature transaction onto the array in the Shipment datastructure
  fetchShipment.temperatures.push(transaction);

  //Map contract and check for temperature reading to be in specified range
  if (tempread.celcius < fetchContract.mintemp || tempread.celcius > fetchContract.maxtemp){
    let tempThresholdEvent = factory.newEvent(NS, 'TemperatureThreshold');
    tempThresholdEvent.temperature = tempread.celcius;
    tempThresholdEvent.latitude = tempread.latitude;
    tempThresholdEvent.longitude = tempread.longitude;
    tempThresholdEvent.readingtime = tempread.readingtime;
    tempThresholdEvent.message = "Temperature range violation! for Shipment $
{fetchShipment.shipmentID}";
    tempThresholdEvent.shipment.shipmentID = fetchShipment.shipmentID;

    emit(tempThesholdEvent);

  }

  //Saving asset registry over blockchain state storage
  return shipmentRegistry.update(fetchShipment);

}




/**
 * Update acceleration readings for an asset
 * @param {org.reliance.network.AccelerationReading} accread
 * @transaction
 */

async function accelerationReading(accread){
```

```
  // Declare global variables and get factory class
  const factory = getFactory();
  const NS = 'org.reliance.network';

  //Fetching contract and shipment for which the acceleration data is to be captured
  let contractRegistry = await getAssetRegistry('org.reliance.network.Contract');
  let shipmentRegistry = await getAssetRegistry('org.reliance.network.Shipment');
  let fetchContract = await contractRegistry.get('accread.shipment.contract.contractID');
  let fetchShipment = await shipmentRegistry.get('accread.shipment.shipmentID');

  //Creating new transaction to capture the data and push into the array in Shipment asset data
structure
  let transaction =factory.newTransaction(NS, 'AccelerationReading');

  transaction.accelerationx = accread.accelerationx;
  transaction.accelerationy = accread.accelerationy;
  transaction.accelerationz = accread.accelerationz;
  transaction.latitude = accread.latitude;
  transaction.longitude = accread.longitude;
  transaction.readingtime = accread.readingtime;

  //Push the acceleration transaction onto the array in the Shipment datastructure
  fetchShipment.accelerations.push(transaction);

  //Map contract and check for acceleration reading to be less than maximum allowed - specified
in the contract

  if(transaction.acclerationx > fetchContract.maxacceleration ||
     transaction.accelerationy > fetchContract.maxacceleration ||
     transaction.accelerationz > fetchContract.maxacceleration ) {

    let accThresholdEvent = factory.newEvent(NS, 'AccelerationThreshold');
    accThresholdEvent.accelerationx = accread.accelerationx;
    accThresholdEvent.accelerationy = accread.accelerationy;
    accThresholdEvent.accelerationz = accread.accelerationz;
    accThresholdEvent.latitude = accread.latitude;
    accThresholdEvent.longitude = accread.longitude;
    accThresholdEvent.readingtime = accread.readingtime;
    accThresholdEvent.shipment.shipmentID = fetchShipment.shipmentID;
    accThresholdEvent.message = "OverAcceleration detected! Check Shipment $
{fetchShipment.shipmentID} for damages!";

    emit(accThesholdEvent);

  }

  //Saving asset registry over blockchain state storage
  return shipmentRegistry.update(fetchShipment);

}




/**
 * Update gps readings for an asset
 * @param {org.reliance.network.GPSReading} gpspread
 * @transaction
 */
```

```
async function GPSReading(gpsread){

  // Declare global variables and get factory class
  const factory = getFactory();
  const NS = 'org.reliance.network';


  //Fetching contract and shipment for which the acceleration data is to be captured
  let contractRegistry = await getAssetRegistry('org.reliance.network.Contract');
  let shipmentRegistry = await getAssetRegistry('org.reliance.network.Shipment');
  let fetchContract = await contractRegistry.get('gpsread.shipment.contract.contractID');
  let fetchShipment = await shipmentRegistry.get('gpsread.shipment.shipmentID');

  //Creating new transaction to capture the data and push into the array in Shipment asset data
structure
  let transaction =factory.newTransaction(NS, 'GPSReading');

  transaction.latitude = gpsread.latitude;
  transaction.latitudedirection = gpsread.latitudedirection;
  transaction.longitude = gpsread.longitude;
  transaction.longitudedirection = gpsread.longitudedirection;
  transaction.readingtime = gpsread.readingtime;
  transaction.readingdate = gpsread.readingdate;

  //Push the gps transaction onto the array in the Shipment datastructure
  fetchShipment.gpsreadings.push(transaction);

  //check if shipment reached destination port

  if(transaction.latitude == fetchShipment.importer.location.latitude &&
     fetchShipment.importer.location.longitude == transaction.longitude){
     let inPortEvent = factory.newEvent(NS, 'ShipmentInPort');
     inPortEvent.shipment.shipmentID = fetchShipment.shipmentID;
     inPortEvent.shipment.status = "arrived";
     inPortEvent.message = " Your shipment ${fetchShipment.shipmentID} reached destination
port. Request you to      collect the same.";

  emit(inPortEvent);

 }

  //Saving asset registry over blockchain state storage
  return shipmentRegistry.update(fetchShipment);

}



/**
 * Shipment received business logic
 * @param {org.reliance.network.ShipmentReceived} received
 * @transaction
 */

 async function ShipmentReceived(received){

  // Declare global variables and get factory class
  const factory = getFactory();
  const NS = 'org.reliance.network';
```

```
  //Fetching contract and shipment for which the acceleration data is to be captured
  let contractRegistry = await getAssetRegistry('org.reliance.network.Contract');
  let shipmentRegistry = await getAssetRegistry('org.reliance.network.Shipment');
  let fetchContract = await contractRegistry.get('received.shipment.contract.contractID');
  let fetchShipment = await shipmentRegistry.get('received.shipment.shipmentID');
  fetchShipment.status = arrived;

  // Calculate actual total payout as per the contract costing
  let totalPayout = fetchShipment.units * fetchContract.unitprice;


  // Penalty considerations
  let now = new Date();
   if(now > fetchContract.arrival){
     totalPayout = 0;
   } else{
     let totalPenalty = fetchShipments.uints * calculatePenaltyFactor(fetchShipment.temperatures,
fetchShipment.accelerations);
     totalPayout = totalPayout - totalPenalty;
   }


  //function to calculate penaltyFactor for temperature and acceleration violations

  function calculatePenaltyFactor(tempArray,accArray){
    let tempviolations = 0, accviolations = 0;
    for (let i = 0 ; i < tempArray.length ; i ++){
      if(tempArray[i] < fetchContract.minTemp || tempArray[i] > fetchContract.maxtemp){
        tempviolations++ ;
      }
    }
    for(let j = 0 ; j < accArray.length ; j++){
      if(accArray[i] > fetchContract.maxacceleration){
        accviolations++ ;
      }
    }

      let penaltyFactor = (tempviolations * fetchContract.maxpenaltyfactor) + (accviolations *
fetchContract.minpenaltyfactor);

      return penaltyFactor;
    }


  // Updating balances of all participants
  fetchContract.exporter.balance += totalPayout;
  fetchContract.shipper.balance += totalPayout * 0.02;
  fetchContract.importer.balance -= (totalpayout + totalPayout *0.02);

  //Saving participant data over blockchain state storage

  let exporterRegistry = await getAssetRegistry('org.reliance.network.Exporter');
  let importerRegistry = await getAssetRegistry('org.reliance.network.Importer');
  let shipperRegistry = await getAssetRegistry('org.reliance.network.Shipper');

  return exporterRegistry.update(fetchContract.exporter);
  return importerRegistry.update(fetchContract.importer);
  return shipperRegistry.update(fetchContract.shipper);
```

}