DETAILED CHAINCODE ALGORITHM

Algorithm 1 RegisterVehicle(): Registers a new vehicle by the manufacturer.

Require: ctx, vehicleDidNo, manufacturerDidNo, partSupplierDidNo, dealerDidNo, parts
Ensure: Success or Error

```
parsedParts ← JSON.parse(parts)
unitIds ← parsedParts.map(part → part.unitId)
vehicle ← {
      vehicleDidNo : vehicleDidNo,
      manufacturerDidNo : manufacturerDidNo,
      partSupplierDidNo : partSupplierDidNo,
      consumerDidNo : "",
      dealerDidNo : dealerDidNo,
      recyclingFacilityDidNo : "",
      status : "Manufactured",
      complianceStatus : "Pending",

  controllerId : "controller_" + vehicleDidNo,
      parts : parsedParts,
      tokens : 0,
      reputation : 0,
      referenceHash : calculateHash(unitIds)
    }
vehicleBuffer ← Buffer.from(JSON.stringify(vehicle))
await ctx.stub.putState(vehicleDidNo, vehicleBuffer)
return JSON.stringify(vehicle)
```

Algorithm 2 SellVehicle(): Records vehicle sale to a consumer.

Require: ctx, vehicleDidNo, consumerDidNo
Ensure: Success or Error

```
vehicleAsBytes ← await ctx.stub.getState(vehicleDidNo)
if vehicleAsBytes = null or vehicleAsBytes.length = 0 then
  Throw Error (vehicleDidNo + " does not exist")
end if
vehicle ← JSON.parse(vehicleAsBytes.toString())
vehicle.consumerDidNo ← consumerDidNo
vehicle.status ← "Sold"
vehicleBuffer ← Buffer.from(JSON.stringify(vehicle))
await ctx.stub.putState(vehicleDidNo, vehicleBuffer)
return JSON.stringify(vehicle)
```

Algorithm 3 UpdateMaintenance(): Updates vehicle maintenance or modifications by a consumer

Require: ctx, vehicleDidNo, maintenanceDetails
Ensure: Success or Error

```
vehicleAsBytes ← await ctx.stub.getState(vehicleDidNo)
if vehicleAsBytes = null or vehicleAsBytes.length = 0 then
  Throw Error (vehicleDidNo + " does not exist")
end if
vehicle ← JSON.parse(vehicleAsBytes.toString())
vehicle.maintenanceDetails ← maintenanceDetails
vehicleBuffer ← Buffer.from(JSON.stringify(vehicle))
await ctx.stub.putState(vehicleDidNo, vehicleBuffer)
return JSON.stringify(vehicle)
```

Algorithm 4 RequestPartsReplacement(): Updates parts replacement and recalculates the hash.

Require: ctx, vehicleDidNo, partDidNo, newExpireDate
Ensure: Success or Error

```
vehicleAsBytes ← await ctx.stub.getState(vehicleDidNo)
if vehicleAsBytes = null or vehicleAsBytes.length = 0 then
  Throw Error(vehicleDidNo + " does not exist")
end if
vehicle ← JSON.parse(vehicleAsBytes.toString())
part ← vehicle.parts.find(p → p.partDidNo = partDidNo)
if part = null then
  Throw Error("Part " + partDidNo + " does not exist on vehicle " + vehicleDidNo)
end if
part.replacementHistory.push({
  replacedOn: New Date().toISOString(),
  oldExpireDate: part.expireDate}
part.expireDate ← newExpireDate
part.currentStatus ← "Replaced"
unitIds ← vehicle.parts.map(p → p.unitId)
vehicle.referenceHash ← this.calculateHash(unitIds)
vehicleBuffer ← Buffer.from(JSON.stringify(vehicle))
await ctx.stub.putState(vehicleDidNo, vehicleBuffer)
return JSON.stringify(vehicle)
```

Algorithm 5 VerifyUnitIntegrity(): Verifies unit integrity by comparing hashes.

Require: ctx, vehicleDidNo
Ensure: Integrity verification result (Success or Tampering Detected)

```
vehicleAsBytes ← await ctx.stub.getState(vehicleDidNo)
if vehicleAsBytes = null or vehicleAsBytes.length = 0 then
    Throw Error(vehicleDidNo + " does not exist")
end if
vehicle ← JSON.parse(vehicleAsBytes.toString())
unitIds ← vehicle.parts.map(p → p.unitId)
currentHash ← this.calculateHash(unitIds)
if currentHash = vehicle.referenceHash then
    return {message: "Unit " + vehicleDidNo+ " integrity verified.", "No tampering detected.", status: "Verified"}
else
    return {message: "Unit " + vehicleDidNo+ " integrity check failed.", "Possible tampering detected.", status: "Tampering Detected"}
end if
```

Algorithm 6 RewardRecycler(): Adds token rewards to the recycler based on recycling quality.

Require: ctx, vehicleDidNo, recyclingFacilityDidNo, purityLevel
Ensure: Reward distribution (Success or Error)

```
vehicleAsBytes ← await ctx.stub.getState(vehicleDidNo)
if vehicleAsBytes = null or vehicleAsBytes.length = 0 then
    Throw Error(vehicleDidNo + " does not exist")
end if
vehicle ← JSON.parse(vehicleAsBytes.toString())
vehicle.recyclingFacilityDidNo ← recyclingFacilityDidNo
tokensEarned ← 0
reputationIncrease ← 0
if purityLevel ≥ 95 then
    tokensEarned ← 50
    reputationIncrease ← 10
else if purityLevel ≥ 80 then
    tokensEarned ← 30
    reputationIncrease ← 5
else if purityLevel ≥ 60 then
    tokensEarned ← 15
    reputationIncrease ← 2
else
    tokensEarned ← 5
    reputationIncrease ← 1
end if
vehicle.tokens ← vehicle.tokens + tokensEarned
vehicle.reputation ← vehicle.reputation + reputationIncrease
vehicle.status ← "Recycled"
await ctx.stub.putState(vehicleDidNo, Buffer.from(JSON.stringify(vehicle)))
return {message: "Recycler rewarded with " + tokensEarned+
    " tokens for purity level of " + purityLevel + "%",
    vehicleDidNo: vehicle.vehicleDidNo,
    tokens: vehicle.tokens,
    reputation: vehicle.reputation}
```

Algorithm 7 GenerateVehicleDIDs(): Generate Decentralized Identifiers (DIDs) for Vehicles, Units, and Parts

Require: User input for Vehicle DID, Unit Name, and Number of Sub-Units/Parts

Ensure: Structured Data with DIDs and Binding Relationships

```
vehicleDid ← UserInput()
unitName ← UserInput()
unitDid ← GenerateDID(unitName)
unitDid.boundTo ← vehicleDid
numParts ← UserInput()
partsDIDs ← []
for i = 1 to numParts do
    partName ← UserInput()
    partDid ← GenerateDID(partName)
    partDid.boundTo ← unitDid
    Append partDid to partsDIDs
end for
vehicleData ← { vehicle: vehicleDid, unit: unitDid, parts: partsDIDs }
Output vehicleData in JSON format
```