Hyperledger Fabric

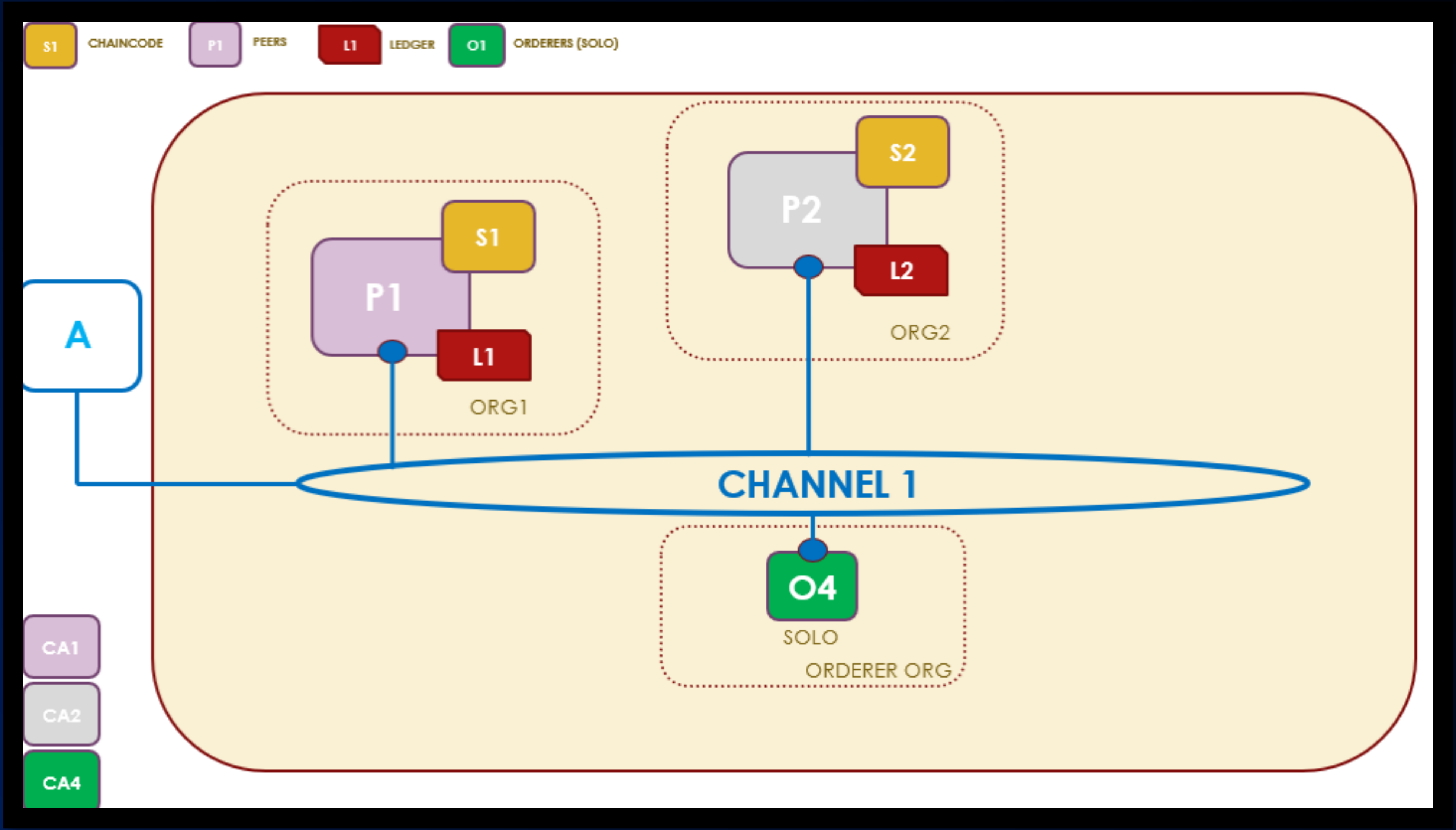
Smart Contract and Client Application

Stefano Avola,

Email: s.avola@cipi.unige.it or s.avola@doc-space.net

Outline

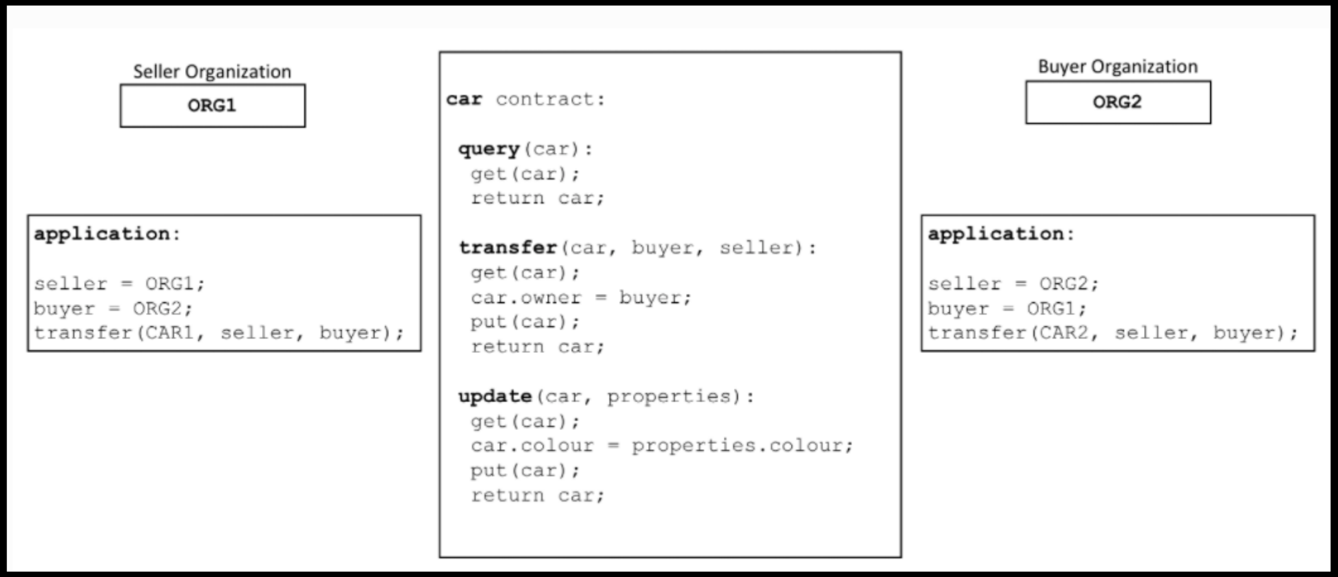
Summing up…


S1
CHAINCODE
P1
PEERS
L1
LEDGER
O1
ORDERERS (SOLO)
S2
P2
L2
ORG2
A
S1
P1
L1
ORG1
CHANNEL 1
O4
SOLO
ORDERER ORG
CA1
CA2
CA4

Smart Contract

Smart Contract

- It defines
  - The (business) logic of the blockchain network,
  - Different business object (asset) states and manages processes that make the state transit.
  - Key data shared among different organizations that join the blockchain network.
- It supports Go, Java, Javascript and Typescript



```
application:

seller = ORG1;
buyer = ORG2;
transfer(CAR1, seller, buyer);
```

```
car contract:

 query(car):
  get(car);
  return car;

 transfer(car, buyer, seller):
  get(car);
  car.owner = buyer;
  put(car);
  return car;

 update(car, properties):
  get(car);
  car.colour = properties.colour;
  put(car);
  return car;
```

```
application:

seller = ORG2;
buyer = ORG1;
transfer(CAR2, seller, buyer);
```

Smart Contract - How to

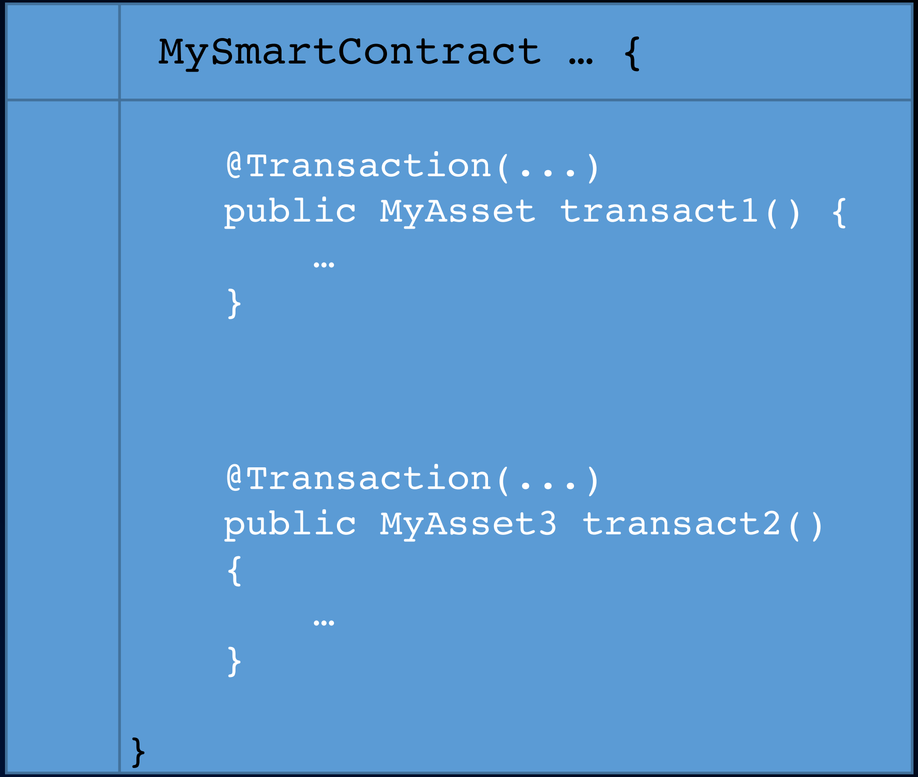
- Contract

```
@Contract(…)
@Default
public final class
MySmartContract implements
ContractInterface {
…
}
```

- Documentation:
- https://hyperledger-fabric.readthedocs.io/en/latest/deploy_chaincode.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/

Smart Contract - How to

- Contract
- Transaction

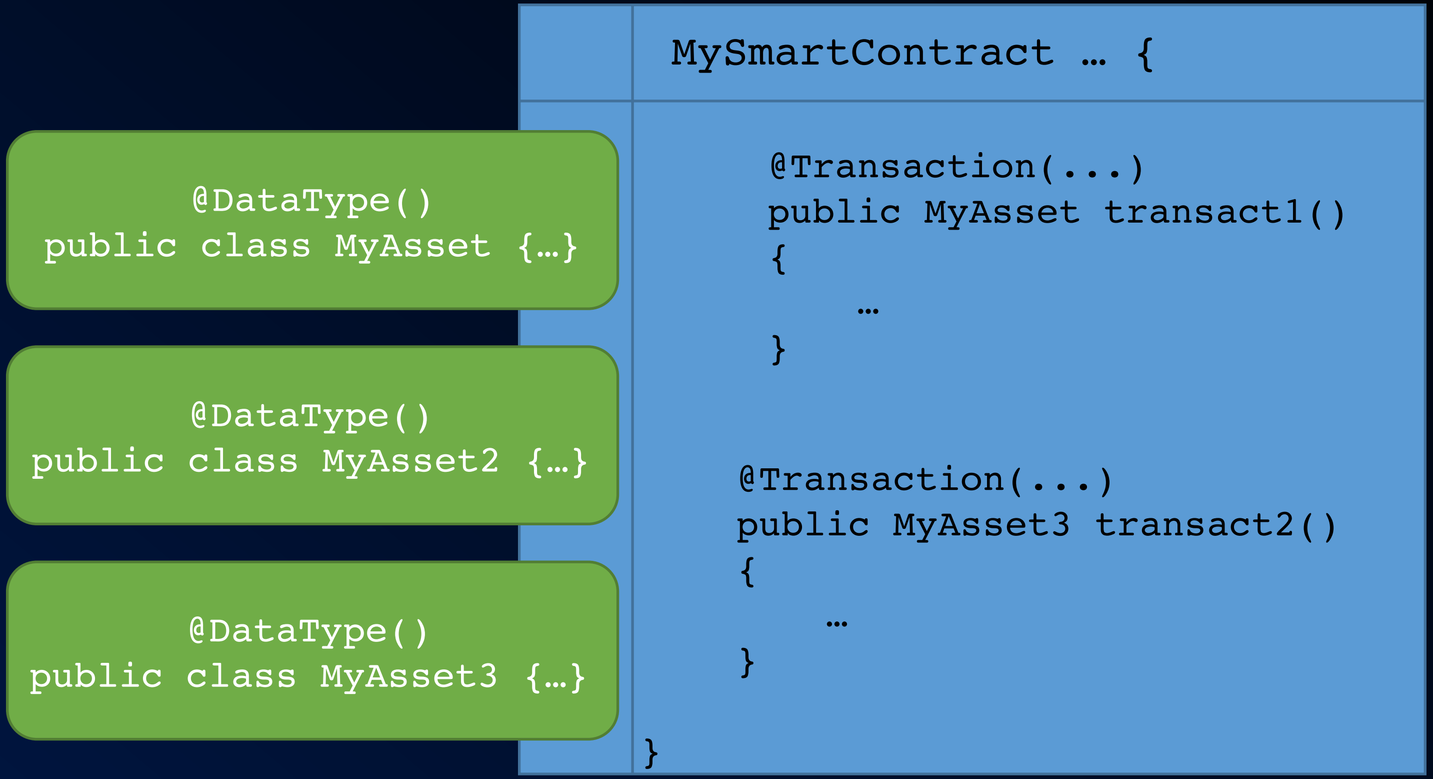


```
MySmartContract … {

    @Transaction(...)
    public MyAsset transact1() {
        …
    }


    @Transaction(...)
    public MyAsset3 transact2()
    {
        …
    }

}
```

- Documentation :
- https://hyperledger-fabric.readthedocs.io/en/latest/deploy_chaincode.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/

Smart Contract - How to

- Contract
- Transaction
- DataType



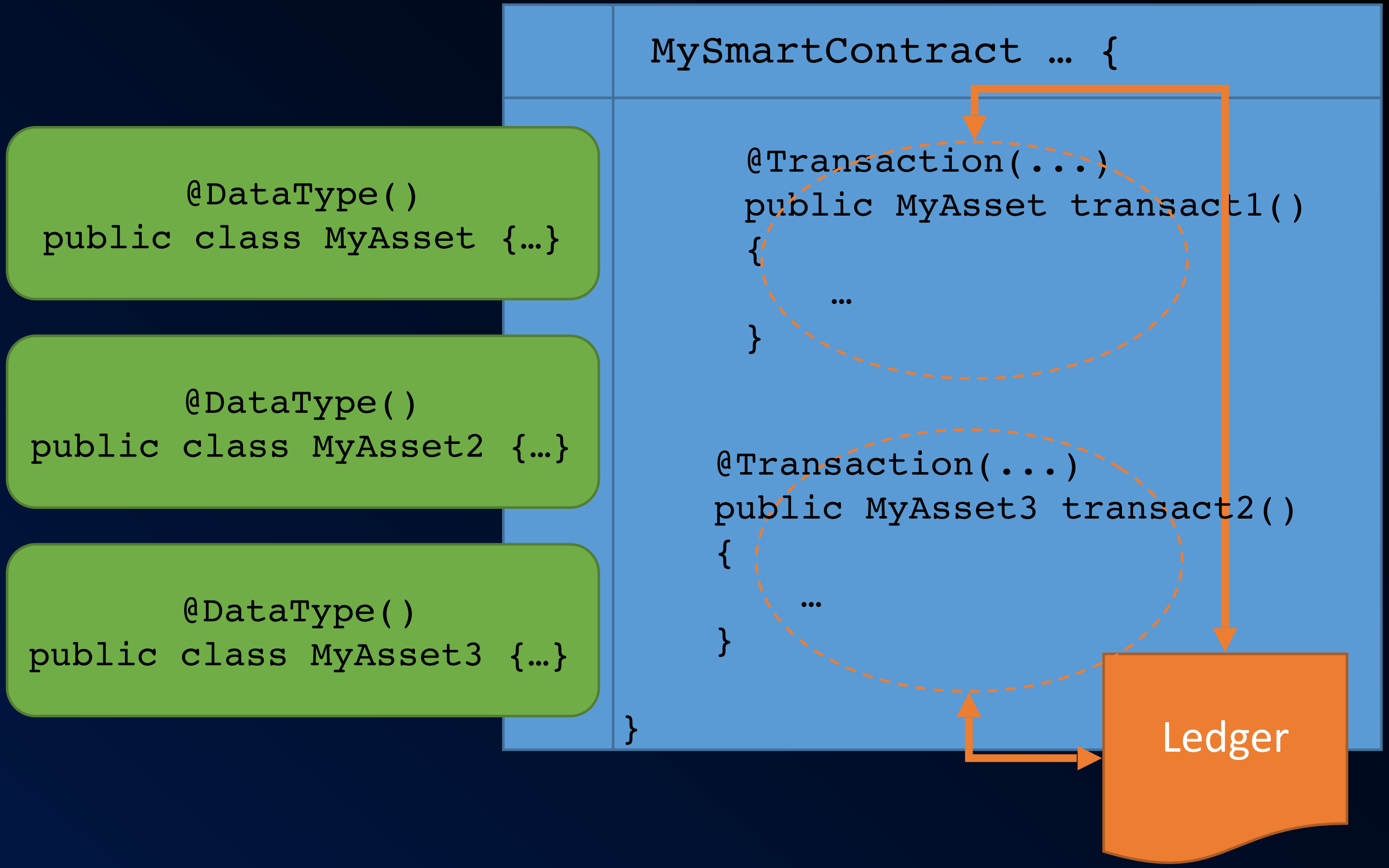
```
@DataType()
public class MyAsset {…}

@DataType()
public class MyAsset2 {…}

@DataType()
public class MyAsset3 {…}
```

```
MySmartContract … {

    @Transaction(...)
    public MyAsset transact1()
    {
        …
    }

    @Transaction(...)
    public MyAsset3 transact2()
    {
        …
    }
}
```

- Documentation :
- https://hyperledger-fabric.readthedocs.io/en/latest/deploy_chaincode.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/

Smart Contract - How to

- Contract
- Transaction
- DataType
- Access to the Ledger



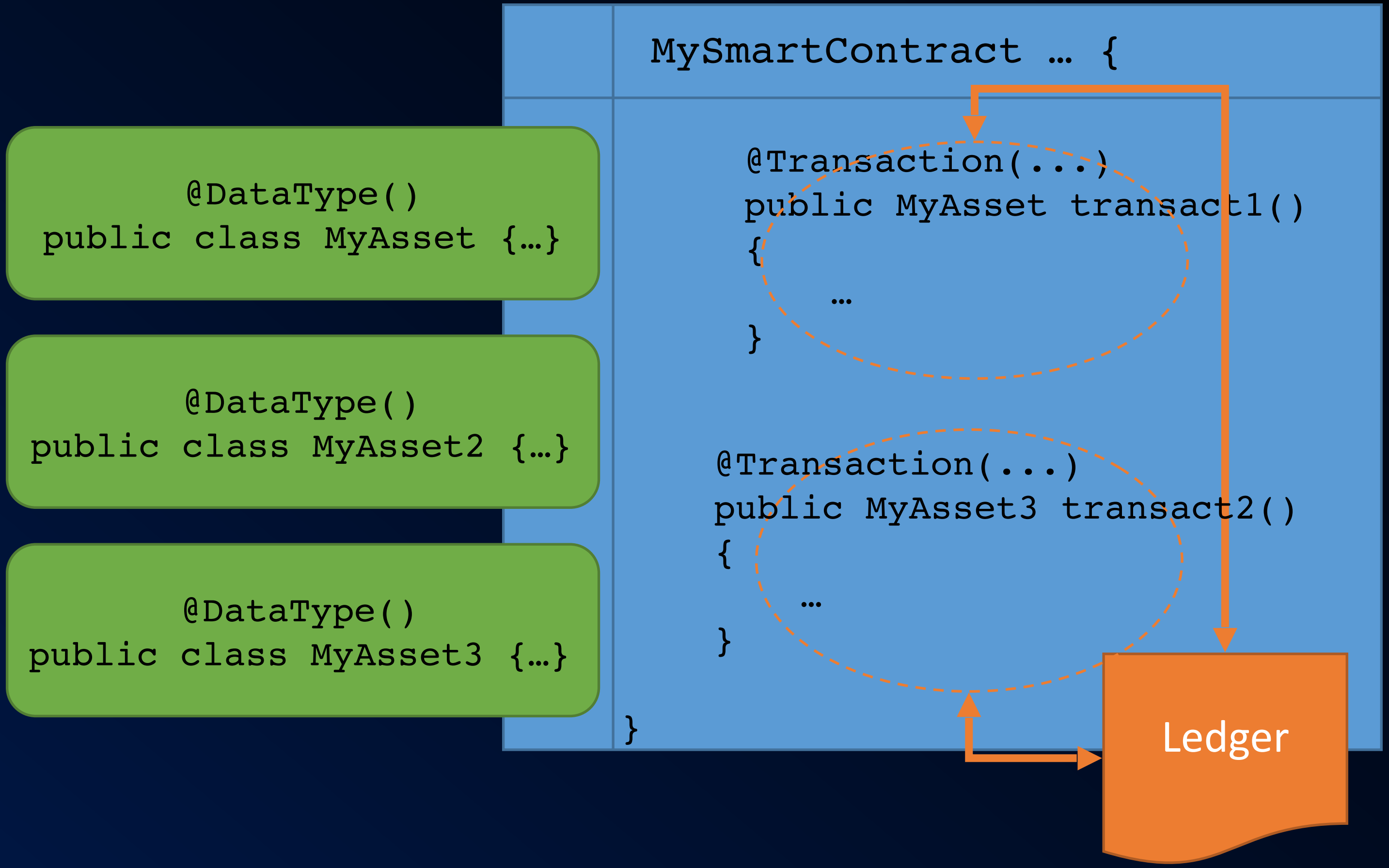
```
@DataType()
public class MyAsset {…}

@DataType()
public class MyAsset2 {…}

@DataType()
public class MyAsset3 {…}
```

```
MySmartContract … {

    @Transaction(...)
    public MyAsset transact1()
    {
        …
    }

    @Transaction(...)
    public MyAsset3 transact2()
    {
        …
    }
}
```

- Documentation :
- https://hyperledger-fabric.readthedocs.io/en/latest/deploy_chaincode.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/

Smart Contract - How to

- Contract
- Transaction
- DataType
- Access to the Ledger
- … a sort of RPC



```
@DataType()
public class MyAsset {…}

@DataType()
public class MyAsset2 {…}

@DataType()
public class MyAsset3 {…}

MySmartContract … {

    @Transaction(...)
    public MyAsset transact1()
    {
        …
    }

    @Transaction(...)
    public MyAsset3 transact2()
    {
        …
    }
}
```

- Documentation :
- https://hyperledger-fabric.readthedocs.io/en/latest/deploy_chaincode.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/

Smart Contract - How to - Contract

`@Contract(…)`

- Class level annotation defining the class as a Smart Contract
- You have to provide the name of the Smart Contract (and also to the Java project)
- You can add additional information:
  - Description, version, contact methods…

`@Default`

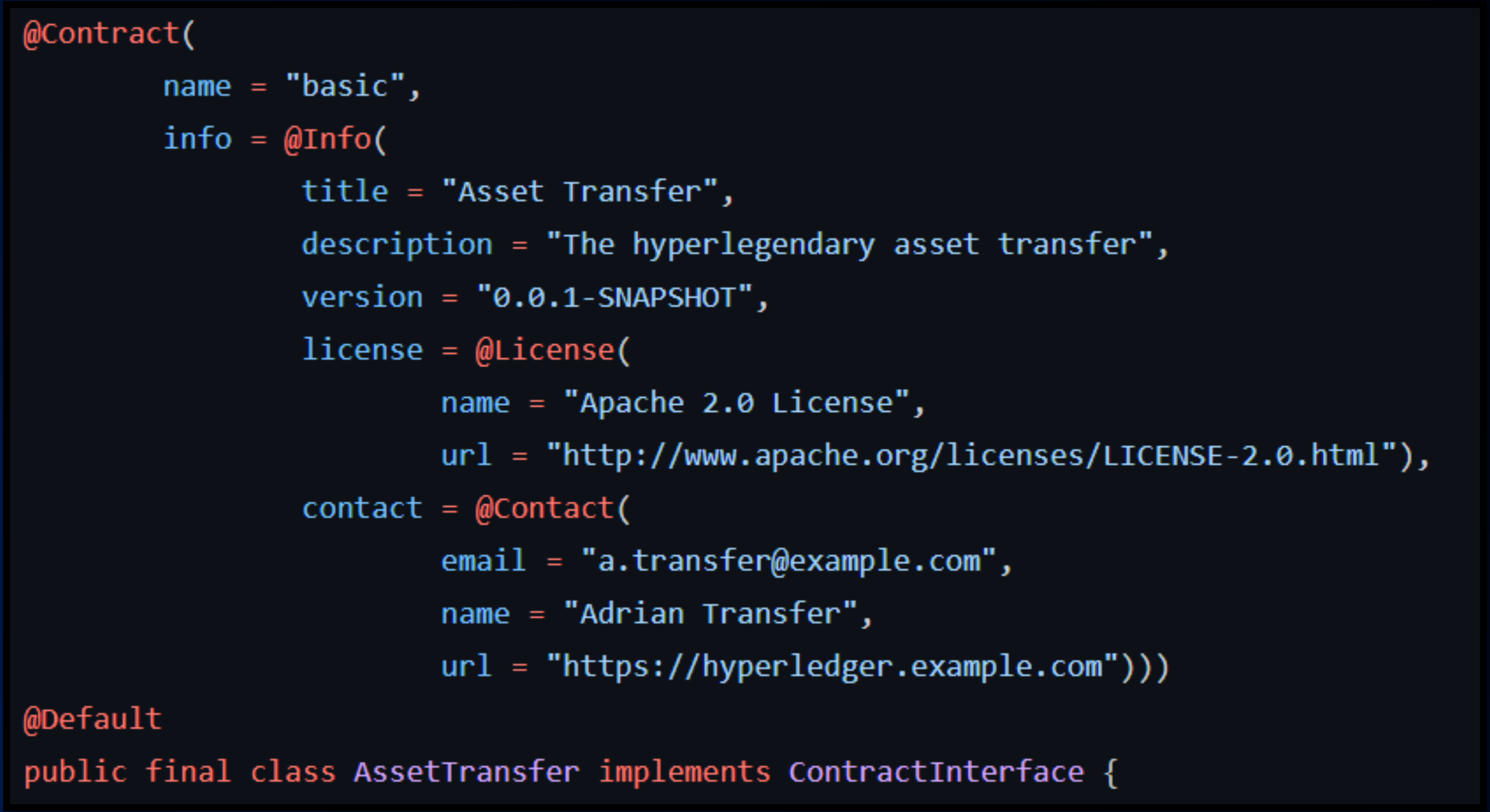
- It defines the annotated class is the default contract class. It is useful in chaincodes having more than one contract class.

`ContractInterface`

- Interface must be implemented by any Contract, because it has methods which have default implementations for Contracts

- Documentation :
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/contract/annotation/Contract.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/contract/package-summary.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/contract/ContractInterface.html

Smart Contract - How to - Contract



```java
@Contract(
        name = "basic",
        info = @Info(
                title = "Asset Transfer",
                description = "The hyperlegendary asset transfer",
                version = "0.0.1-SNAPSHOT",
                license = @License(
                        name = "Apache 2.0 License",
                        url = "http://www.apache.org/licenses/LICENSE-2.0.html"),
                contact = @Contact(
                        email = "a.transfer@example.com",
                        name = "Adrian Transfer",
                        url = "https://hyperledger.example.com")))
@Default
public final class AssetTransfer implements ContractInterface {
```

- Documentation :
- https://github.com/hyperledger/fabric-samples/blob/main/asset-transfer-basic/chaincode-java/src/main/java/org/hyperledger/fabric/samples/assettransfer/AssetTransfer.java

Smart Contract - How to - Transaction

`@Transaction(…)`

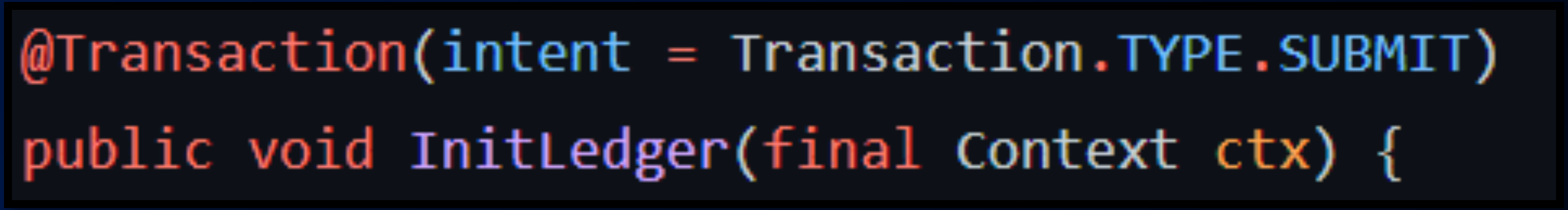
- Method level annotation specifying that the method is a callable transaction from the client application
- It has an element «`intent`» which defines the transaction type:
  - Submit («put»)
  - Evaluate («get»)

`Context`

- It represents the *context*/environment where the transaction is called. Once the transaction is ended, the context is cancelled.
- It provides access to the APIs for the interaction with the Ledger..but not only that…

- Documentation :
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/contract/annotation/Transaction.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/contract/annotation/Transaction.TYPE.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/contract/Context.html

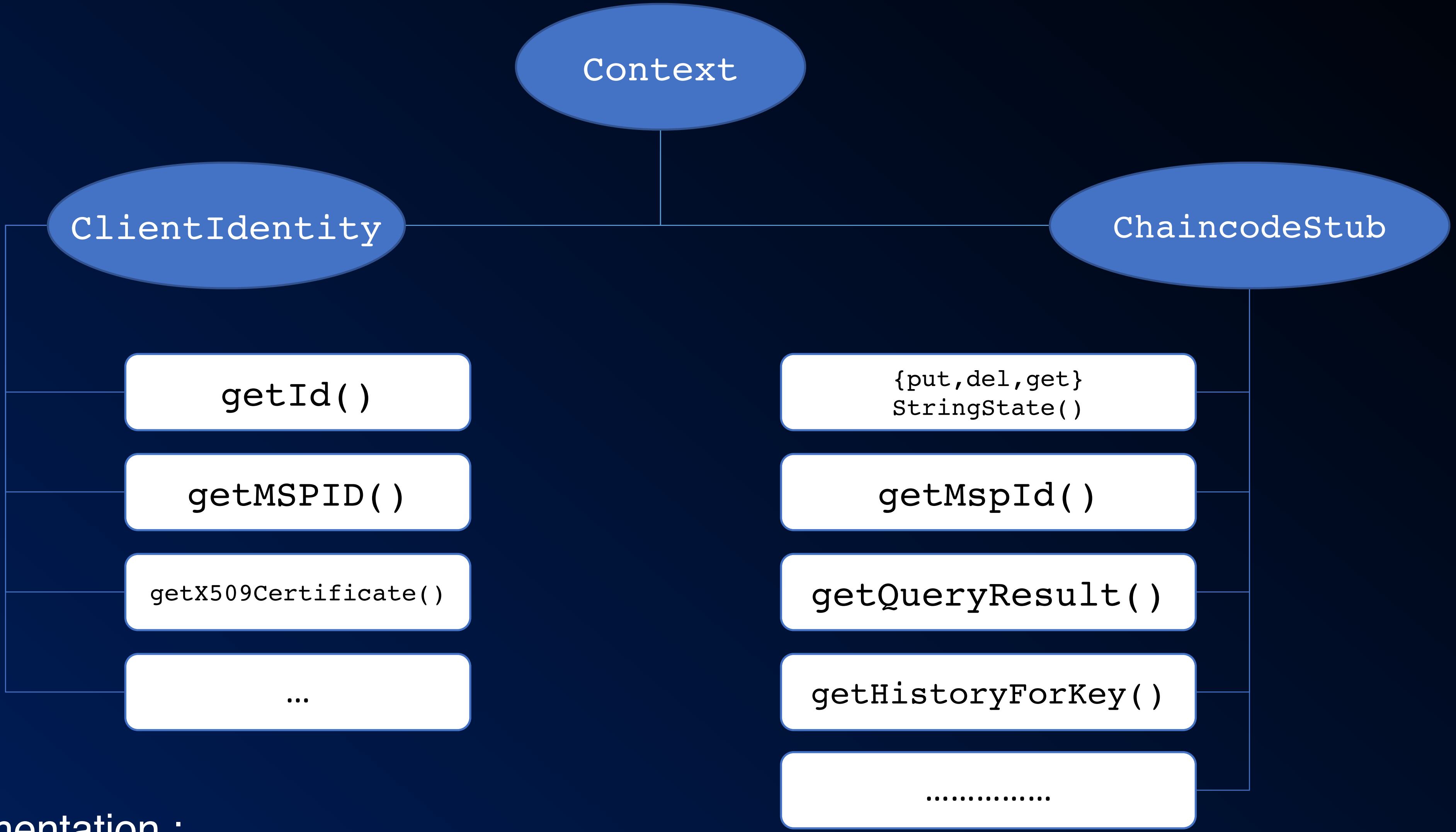
Smart Contract - How to - Transaction



```
@Transaction(intent = Transaction.TYPE.SUBMIT)
public void InitLedger(final Context ctx) {
```

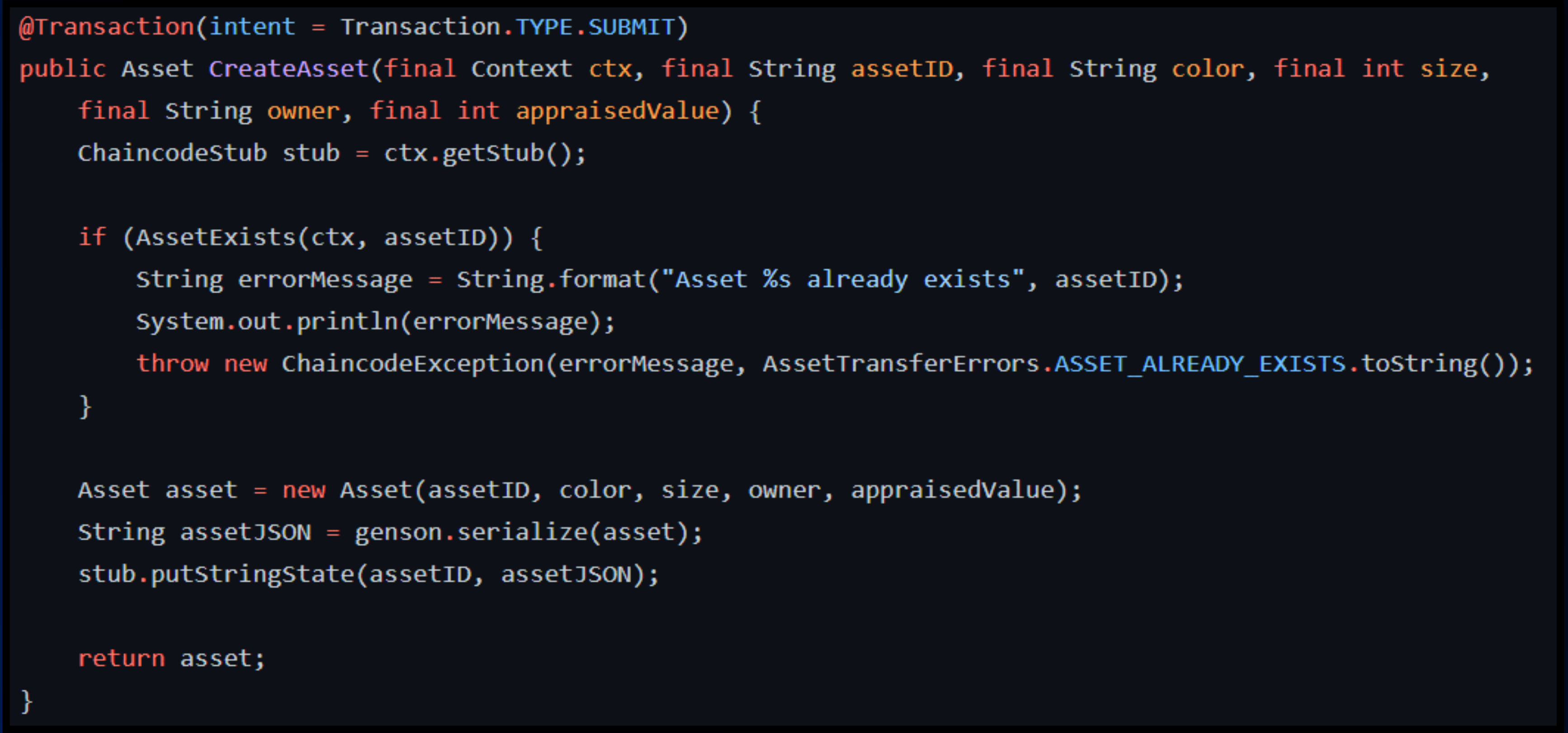
- Documentation :
- https://github.com/hyperledger/fabric-samples/blob/main/asset-transfer-basic/chaincode-java/src/main/java/org/hyperledger/fabric/samples/assettransfer/AssetTransfer.java

Smart Contract - How to - Transaction - Context

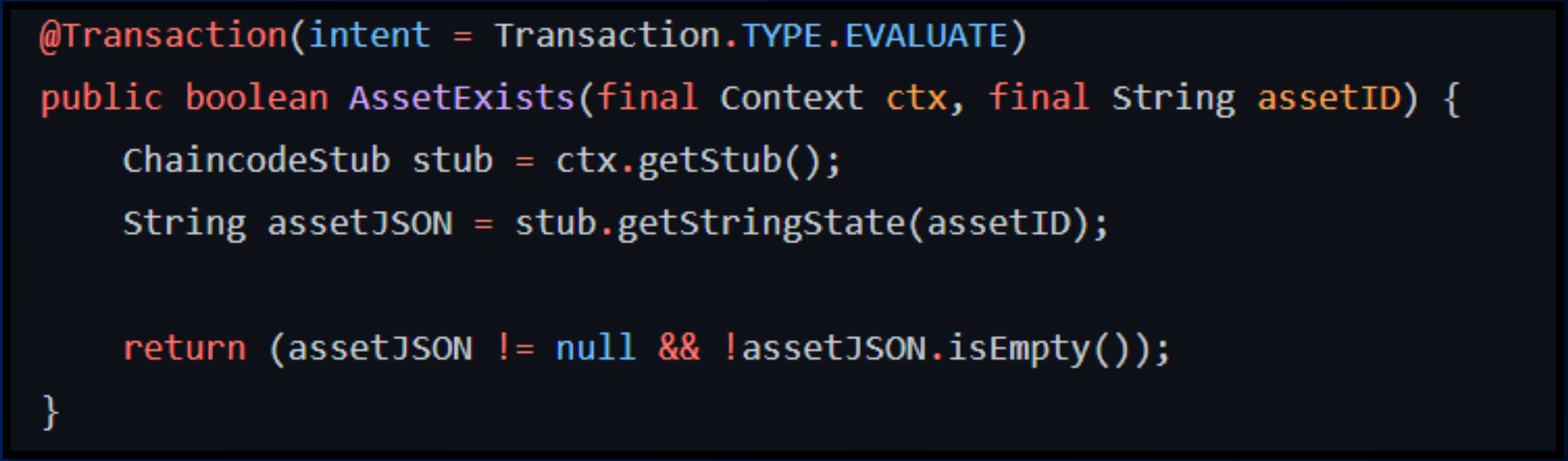


- Documentation :
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/contract/Context.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/shim/ChaincodeStub.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/contract/ClientIdentity.html

Smart Contract - How to - Transaction



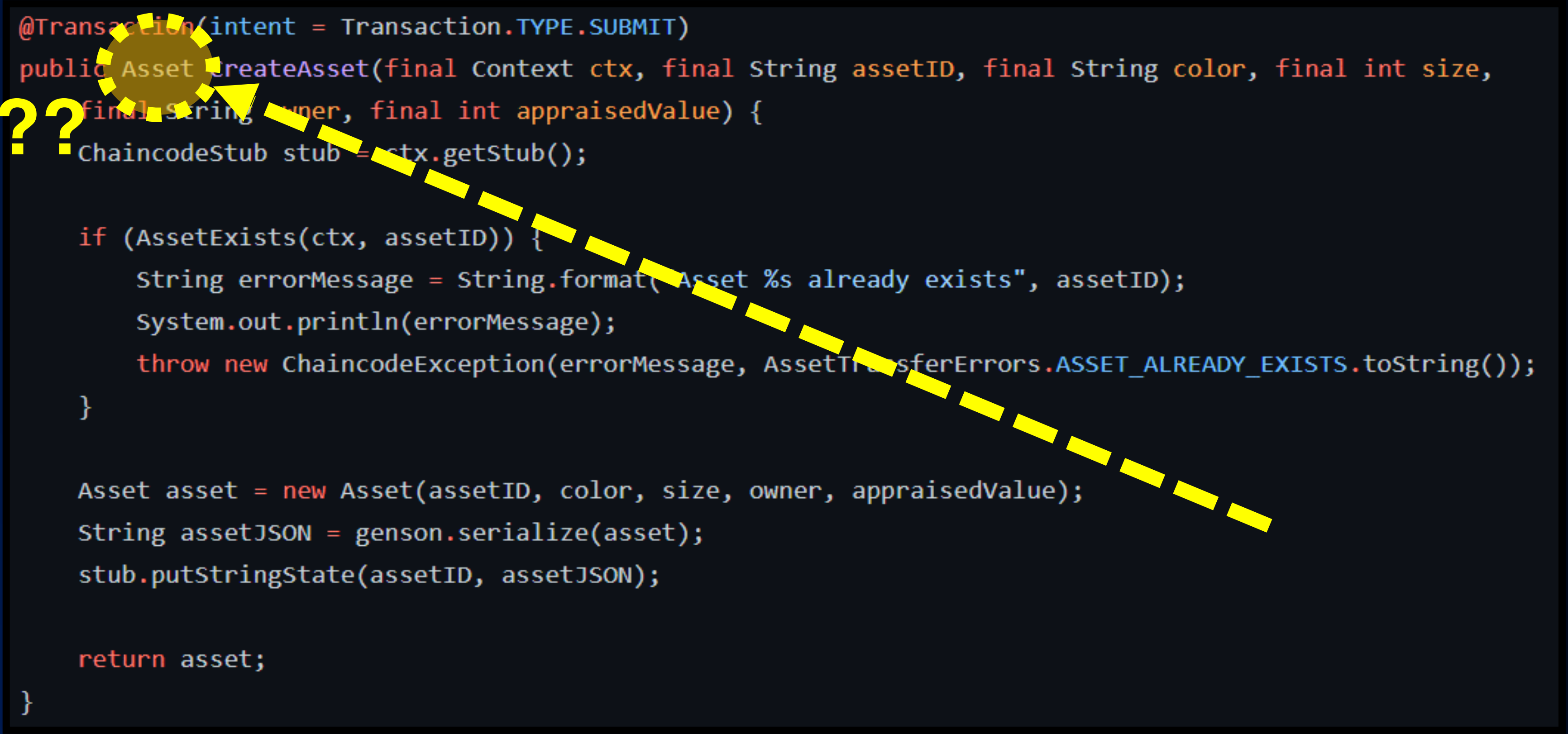
```java
@Transaction(intent = Transaction.TYPE.SUBMIT)
public Asset CreateAsset(final Context ctx, final String assetID, final String color, final int size,
    final String owner, final int appraisedValue) {
    ChaincodeStub stub = ctx.getStub();

    if (AssetExists(ctx, assetID)) {
        String errorMessage = String.format("Asset %s already exists", assetID);
        System.out.println(errorMessage);
        throw new ChaincodeException(errorMessage, AssetTransferErrors.ASSET_ALREADY_EXISTS.toString());
    }

    Asset asset = new Asset(assetID, color, size, owner, appraisedValue);
    String assetJSON = genson.serialize(asset);
    stub.putStringState(assetID, assetJSON);

    return asset;
}
```



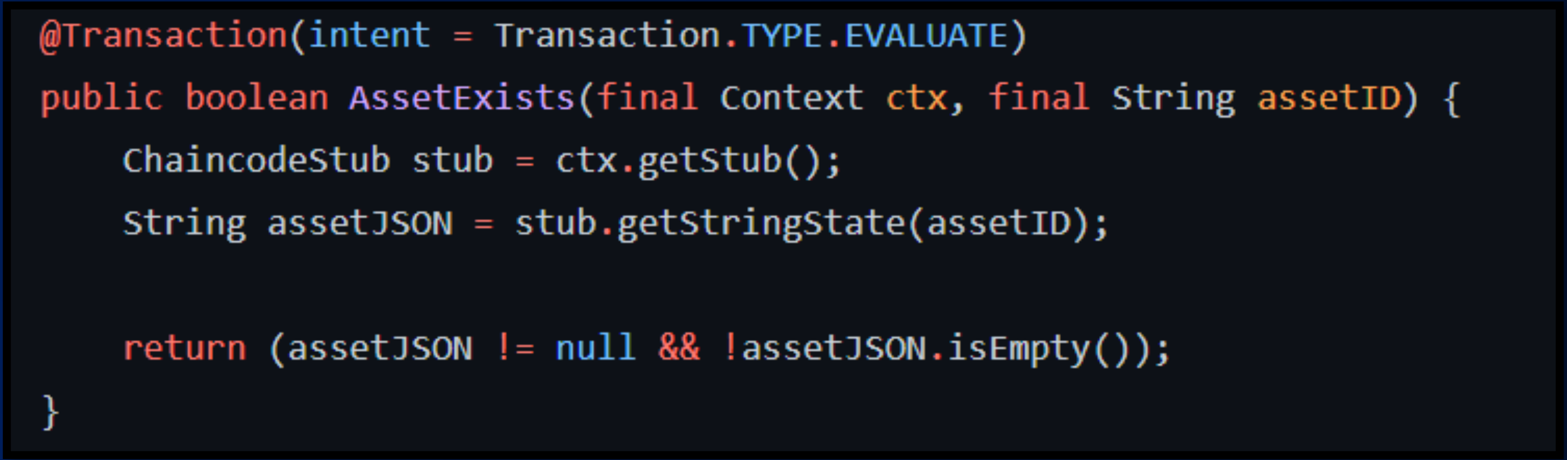
```java
@Transaction(intent = Transaction.TYPE.EVALUATE)
public boolean AssetExists(final Context ctx, final String assetID) {
    ChaincodeStub stub = ctx.getStub();
    String assetJSON = stub.getStringState(assetID);

    return (assetJSON != null && !assetJSON.isEmpty());
}
```

- Documentation :
- https://github.com/hyperledger/fabric-samples/blob/main/asset-transfer-basic/chaincode-java/src/main/java/org/hyperledger/fabric/samples/assettransfer/AssetTransfer.java

Smart Contract - How to - Transaction



```
@Transaction(intent = Transaction.TYPE.SUBMIT)
public Asset CreateAsset(final Context ctx, final String assetID, final String color, final int size,
    final String owner, final int appraisedValue) {
    ChaincodeStub stub = ctx.getStub();

    if (AssetExists(ctx, assetID)) {
        String errorMessage = String.format("Asset %s already exists", assetID);
        System.out.println(errorMessage);
        throw new ChaincodeException(errorMessage, AssetTransferErrors.ASSET_ALREADY_EXISTS.toString());
    }

    Asset asset = new Asset(assetID, color, size, owner, appraisedValue);
    String assetJSON = genson.serialize(asset);
    stub.putStringState(assetID, assetJSON);

    return asset;
}
```



```
@Transaction(intent = Transaction.TYPE.EVALUATE)
public boolean AssetExists(final Context ctx, final String assetID) {
    ChaincodeStub stub = ctx.getStub();
    String assetJSON = stub.getStringState(assetID);

    return (assetJSON != null && !assetJSON.isEmpty());
}
```

- Documentation :
- https://github.com/hyperledger/fabric-samples/blob/main/asset-transfer-basic/chaincode-java/src/main/java/org/hyperledger/fabric/samples/assettransfer/AssetTransfer.java

Smart Contract - How to - DataType

`@DataType()`

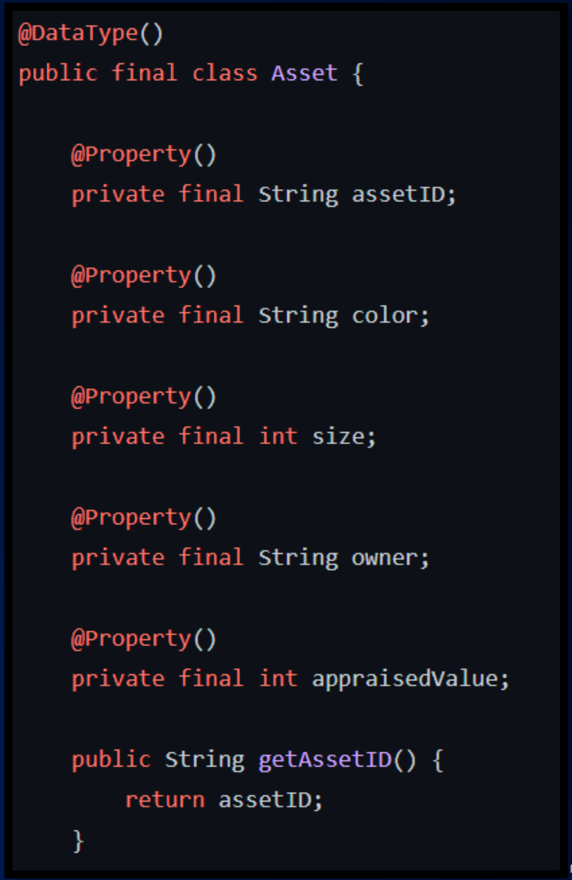
- Class level annotation specifying that the class is one of the types which can be **returned** by or passed as arguments to transactions
- Serialization and deserialization, in 2 ways (not simultaneously!):
  - Genson
    - Constructor with `@JsonProperty(‘’…’’)`, genson’s annotaion (see example)
  - Gson
    - To your liking, usually the serialize() and deserialize() methods are implemented.

`@Property()`

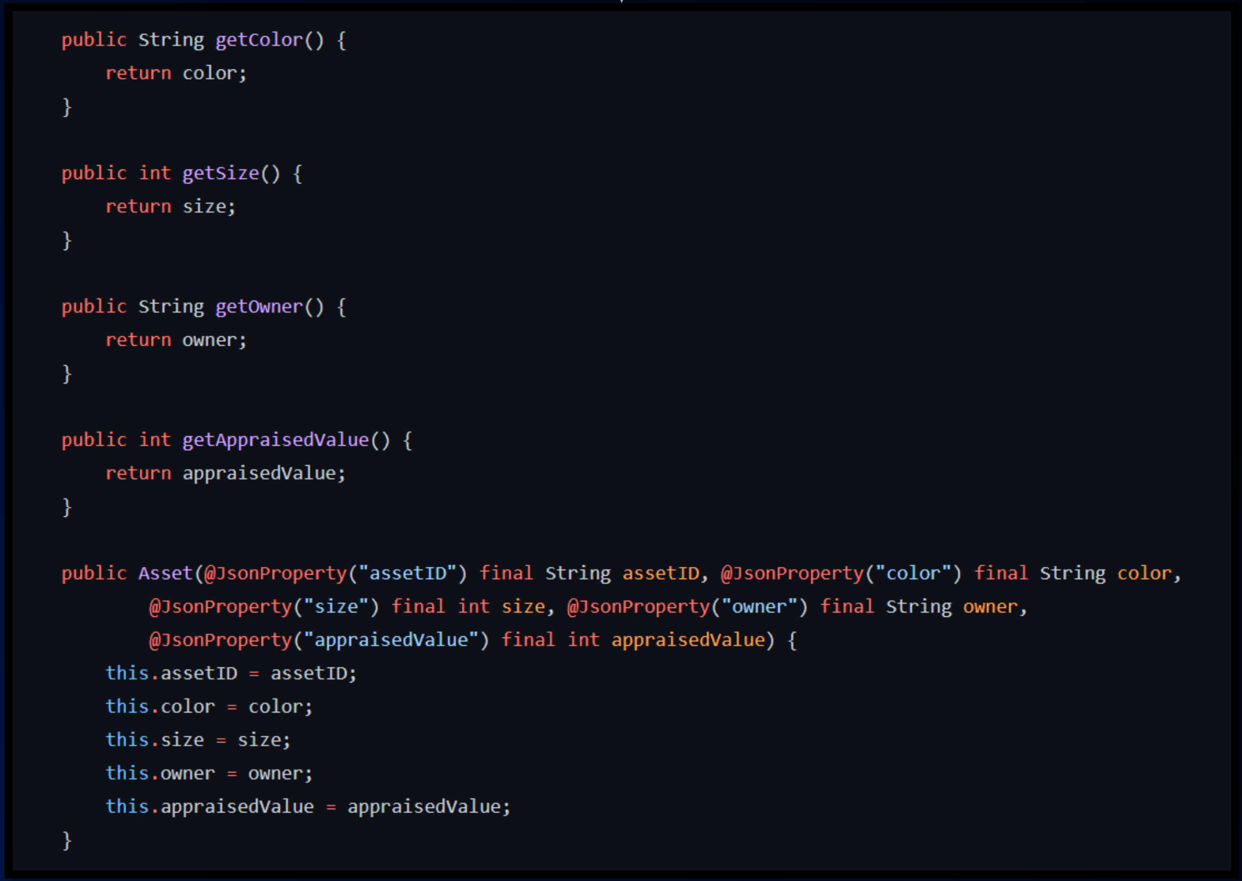
- Field (and parameter) level annotation defining the field (parameter) as a *property* of the class

- Documentation :
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/contract/annotation/DataType.html
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/contract/annotation/Property.html

Smart Contract - How to - DataType



```java
@DataType()
public final class Asset {

    @Property()
    private final String assetID;

    @Property()
    private final String color;

    @Property()
    private final int size;

    @Property()
    private final String owner;

    @Property()
    private final int appraisedValue;

    public String getAssetID() {
        return assetID;
    }
```



```java
public String getColor() {
    return color;
}

public int getSize() {
    return size;
}

public String getOwner() {
    return owner;
}

public int getAppraisedValue() {
    return appraisedValue;
}

public Asset(@JsonProperty("assetID") final String assetID, @JsonProperty("color") final String color,
        @JsonProperty("size") final int size, @JsonProperty("owner") final String owner,
        @JsonProperty("appraisedValue") final int appraisedValue) {
    this.assetID = assetID;
    this.color = color;
    this.size = size;
    this.owner = owner;
    this.appraisedValue = appraisedValue;
}
```

- Documentation :
- https://github.com/hyperledger/fabric-samples/blob/main/asset-transfer-basic/chaincode-java/src/main/java/org/hyperledger/fabric/samples/assettransfer/Asset.java

Smart Contract – How to – Access to the Ledger

`putStringState(String key, String value)`	• It (over-)writes into the Ledger, when the transaction is commited, the value *value* associated with the asset/object with key *key.*
`getStringState(String key)`	• It returns the committed value in the Ledger of the asset with key *key*.
`getStateByRange(String startKey, String endKey)`	• It returns all the existing keys with their related values. Keys are returned in lexicographical (asset1 < asset11 < asset2) order between the key *startkey* (inclusive) and the one *endKey* (exclusive).
`getQueryResult(String query)`	• It returns all the values related to a «rich» query. • The parameter *query* must be in a format supported by CouchDB
`getHistoryForKey(String key)`	• It returns the history of the asset with key *key*, including txID and timestamp.

- Documentation :
- https://hyperledger.github.io/fabric-chaincode-java/main/api/org/hyperledger/fabric/shim/ChaincodeStub.html

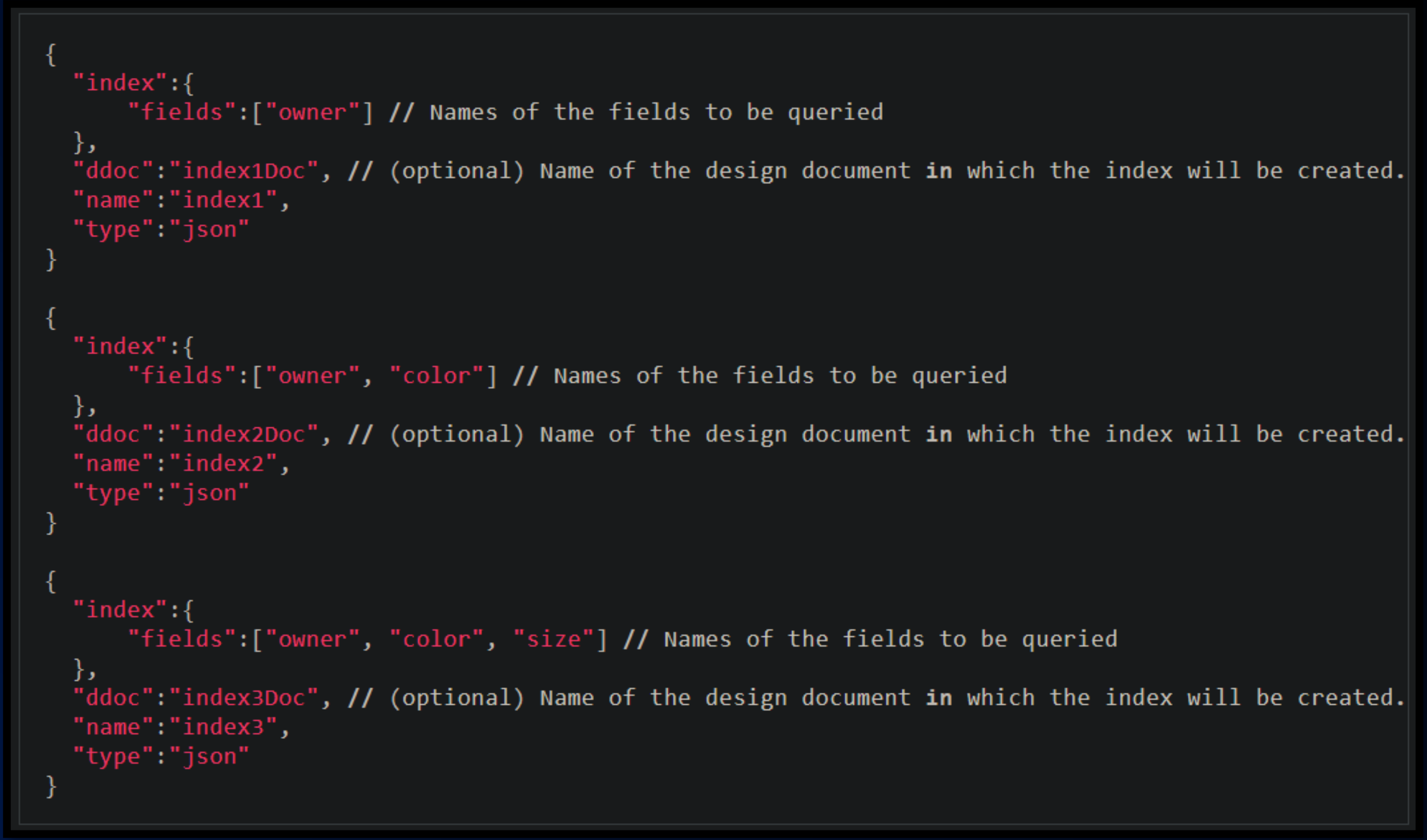
Smart Contract - How to - Access to the Ledger - Indexes (Optional)

- They allow querying the database with «rich» queries, that is querying the database by any fields and not only by the ID.

- 3 information are needed:
  - `Fields`: fields to query
  - `Name`: index's name
  - `Type`: always "json"

- Example of indexes are shown in fig.

- CouchDB provides a UI to test your indexes

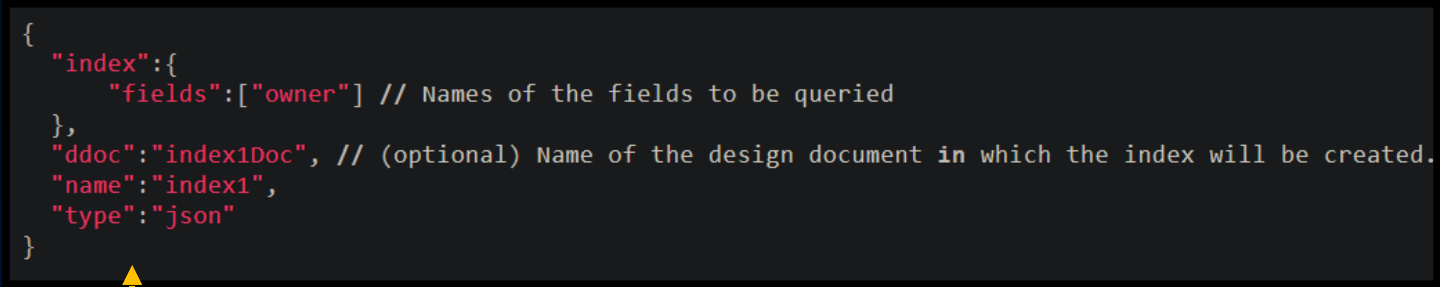


```
{
  "index":{
      "fields":["owner"] // Names of the fields to be queried
  },
  "ddoc":"index1Doc", // (optional) Name of the design document in which the index will be created.
  "name":"index1",
  "type":"json"
}

{
  "index":{
      "fields":["owner", "color"] // Names of the fields to be queried
  },
  "ddoc":"index2Doc", // (optional) Name of the design document in which the index will be created.
  "name":"index2",
  "type":"json"
}

{
  "index":{
      "fields":["owner", "color", "size"] // Names of the fields to be queried
  },
  "ddoc":"index3Doc", // (optional) Name of the design document in which the index will be created.
  "name":"index3",
  "type":"json"
}
```

- Documentation :
- https://hyperledger-fabric.readthedocs.io/en/release-2.4/couchdb_tutorial.html

Smart Contract - How to – Access to the Ledger - Indexes (Optional)

- How to use them in practice?
  1. In the directory containing the chaincode, create a file (e.g.) `index1.json` to the path `META-INF/statedb/couchdb/indexes/`. Such file should be similar to the one in fig.,
  2. Deploy the chaincode, without specifying the existence of the index (see next slide),
  3. In the Smart Contract use `getQueryResult()` with an input as the example shown in fig.



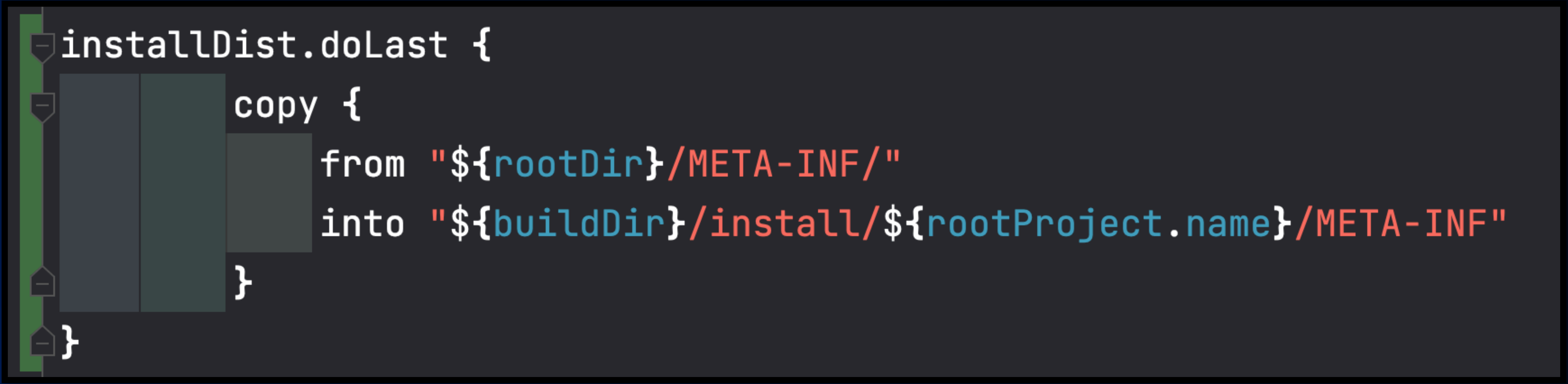
```
{
  "index":{
      "fields":["owner"] // Names of the fields to be queried
  },
  "ddoc":"index1Doc", // (optional) Name of the design document in which the index will be created.
  "name":"index1",
  "type":"json"
}
```

```
"{
  \"selector\":
        {
          \"owner\":\"ABC\"
        },
  \"use_index\":
        [\"_design/index1Doc\",
\"index1\"]
}"
```

- Documentation :
- https://hyperledger-fabric.readthedocs.io/en/release-2.4/couchdb_tutorial.html

Smart Contract - How to – Access to the Ledger - Indexes (Optional)

- NOTE: Moreover, in Java you have to make sure that `META-INF` is under `<chaincode_root_directory>/build/install/<chaincode_name>`.
  - To do that you have to add the lines in figure to the `build.gradle` file located in the chaincode project root:



```
installDist.doLast {
    copy {
        from "${rootDir}/META-INF/"
        into "${buildDir}/install/${rootProject.name}/META-INF"
    }
}
```

  - Otherwise you can use the chainconde template project that you can download from the GitHub link shown on the hands on slide

Notes

- Deployment of an updated version of the `basic` chaincode
  - `./network.sh deployCC -ccn basic -ccp /path/to/chaincode/ -ccl java -ccv` *`new_version`*
  - E.g.:
    - First deployment
      - `./network.sh deployCC -ccn basic -ccp /path/to/chaincode/ -ccl java`
    - Second deployment to the version “`2.0`”
      - `./network.sh deployCC -ccn basic -ccp /path/to/chaincode/ -ccl java -ccv 2.0`
    - Third deployment to the version “`2.1`”
      - `./network.sh deployCC -ccn basic -ccp /path/to/chaincode/ -ccl java -ccv 2.1`
    - And so on..

Notes

- Until now we always used the chaincode "`basic`", if you want to change the name you should
  - Modify the `rootProject.name` in `settings.gradle` (located in the chaincode project root directory)
  - Change the element `name` in `@Contract`
  - Deploy it with `-ccn <new_name>`

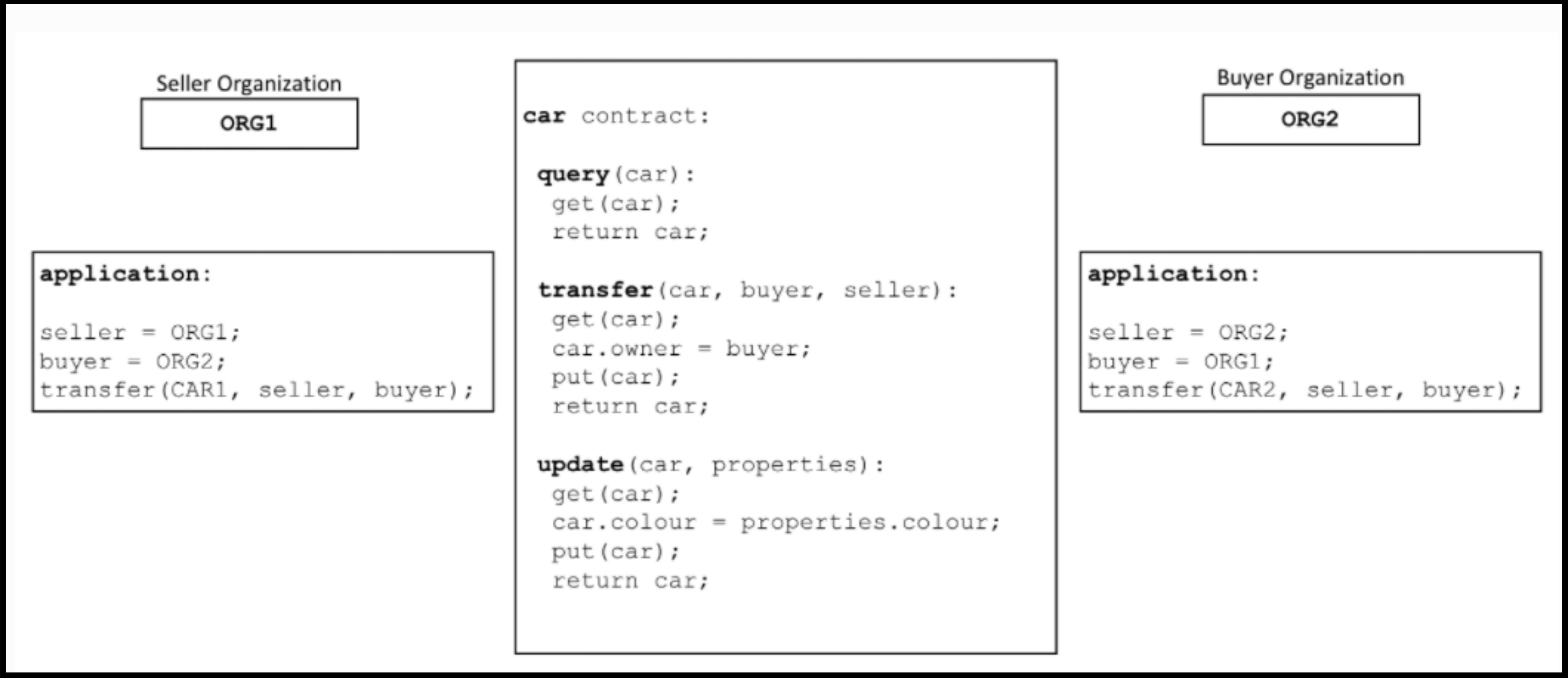
- `build.gradle` (only for the Client Application): modify `javaMainClass = "application.java.MyApp"` with `javaMainClass = "<your_main_class>"`

- CouchDB's changes only returns the last change of an asset

Smart Contract - Hands on

1. (`./network.sh down` or `docker restart $(docker ps -aq)`)
2. Download the template project of a chaincode from https://github.com/avolast/hyperledger-fabric-simple-template
3. Import the project in an IDE as a gradle project
4. Implement a Smart Contract (Car Contract in the next slide)
5. Deploy chaincode
6. Interact through CLI (for the time being). See the `interact_as_org{1,2}_cli.sh` files .
7. Exercise to solve
8. Next lab session: create your own application

Smart Contract - Hands on



```
application:

seller = ORG1;
buyer = ORG2;
transfer(CAR1, seller, buyer);
```

```
car contract:

 query(car):
  get(car);
  return car;

 transfer(car, buyer, seller):
  get(car);
  car.owner = buyer;
  put(car);
  return car;

 update(car, properties):
  get(car);
  car.colour = properties.colour;
  put(car);
  return car;
```

```
application:

seller = ORG2;
buyer = ORG1;
transfer(CAR2, seller, buyer);
```

- We also add the “create”

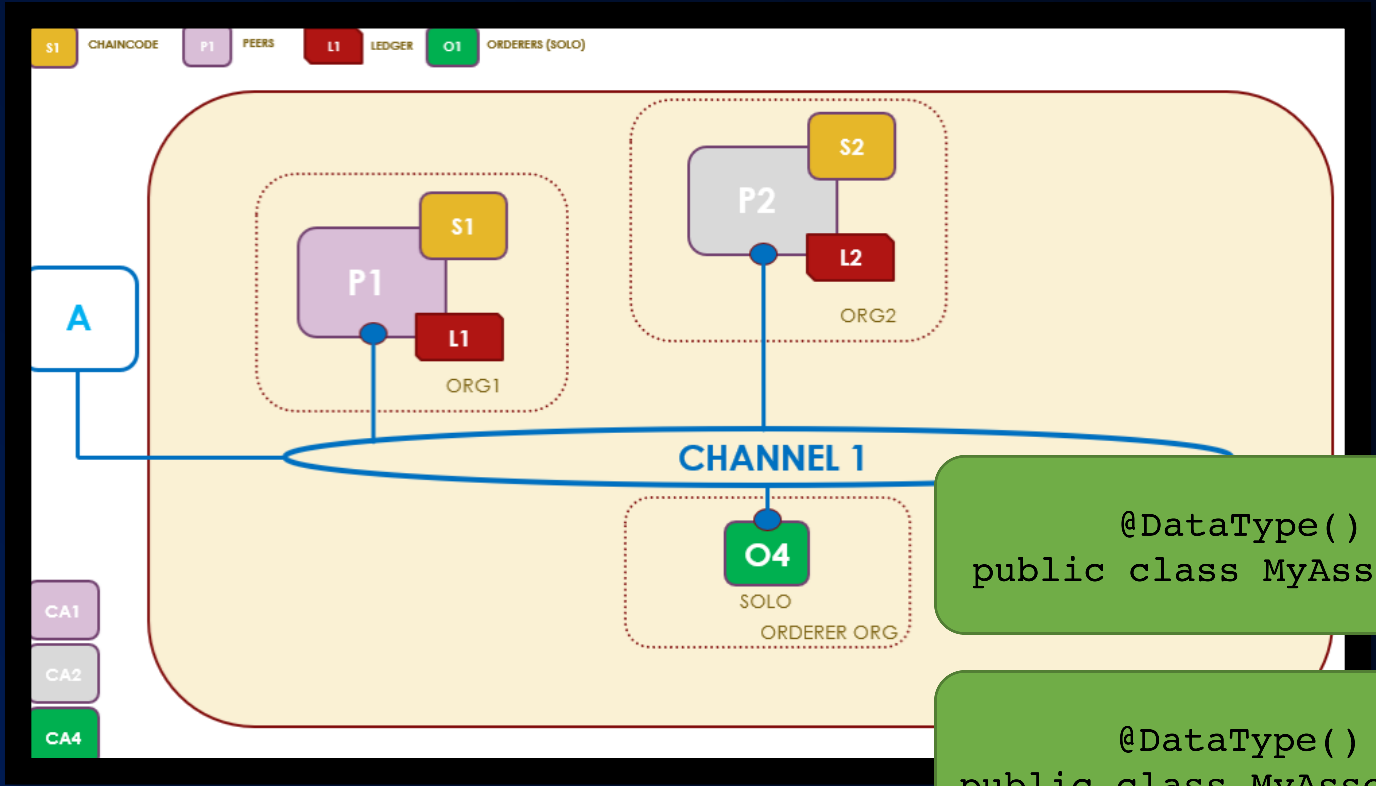
Exercise - Smart Contract

Ownership transfer of documents

- Context
  - In the logistics environment there is a document called «bill of lading». In the paper world it is a single copy document related to goods and it can be passed from hand to hand each time to the new (legitimate) owner, who has the goods ownership.
- In this exercise you assume that the owner is an organization of the channel.
- Implement a Smart Contract such that allows
  - The creation of a bill of lading; by default associating it with the creator organization.
  - The ownership transfer of a bill of lading
    - Verify that the organization that wants to transfer the ownership is the same submitting the transfer transaction.
  - Getting
    - The actual state of a bill of lading,
    - The history of a bill of lading,
    - The actual state of all bills of lading,
    - All bills of lading of a specific organization (optional)
- Use the language you prefer among those allowed.

Client Application

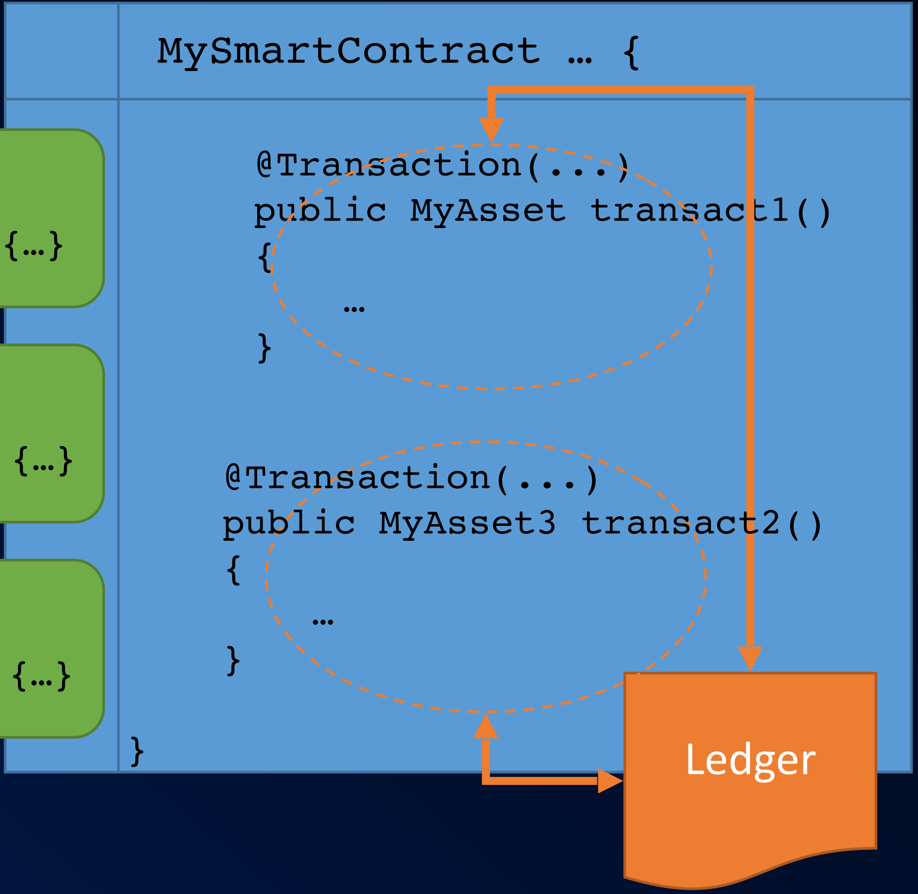
Summing up…



```
@DataType()
public class MyAsset {…}
```

```
@DataType()
public class MyAsset2 {…}
```

```
@DataType()
public class MyAsset3 {…}
```



```
MySmartContract … {

    @Transaction(...)
    public MyAsset transact1()
    {
        …
    }

    @Transaction(...)
    public MyAsset3 transact2()
    {
        …
    }
}
```

Client Application

- It allows a remote interaction with the Smart Contract / Chaincode installed on peers
- It has to connect to an organization’s peer in a secure way
- An application can connect to different peers and interact as a user of an organization
  - Obviously, it can interact as more users even belonging to different organizations, but always one by one…even if, for each tx, it has to ask other peers for approval.
- Transaction flow (see slides on the first laboratory on Hyperledger Fabric)
  - Everything is automatically handled, but you can override the behavior

- Same languages supported by the Smart Contract
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html

Client Application - Requirements

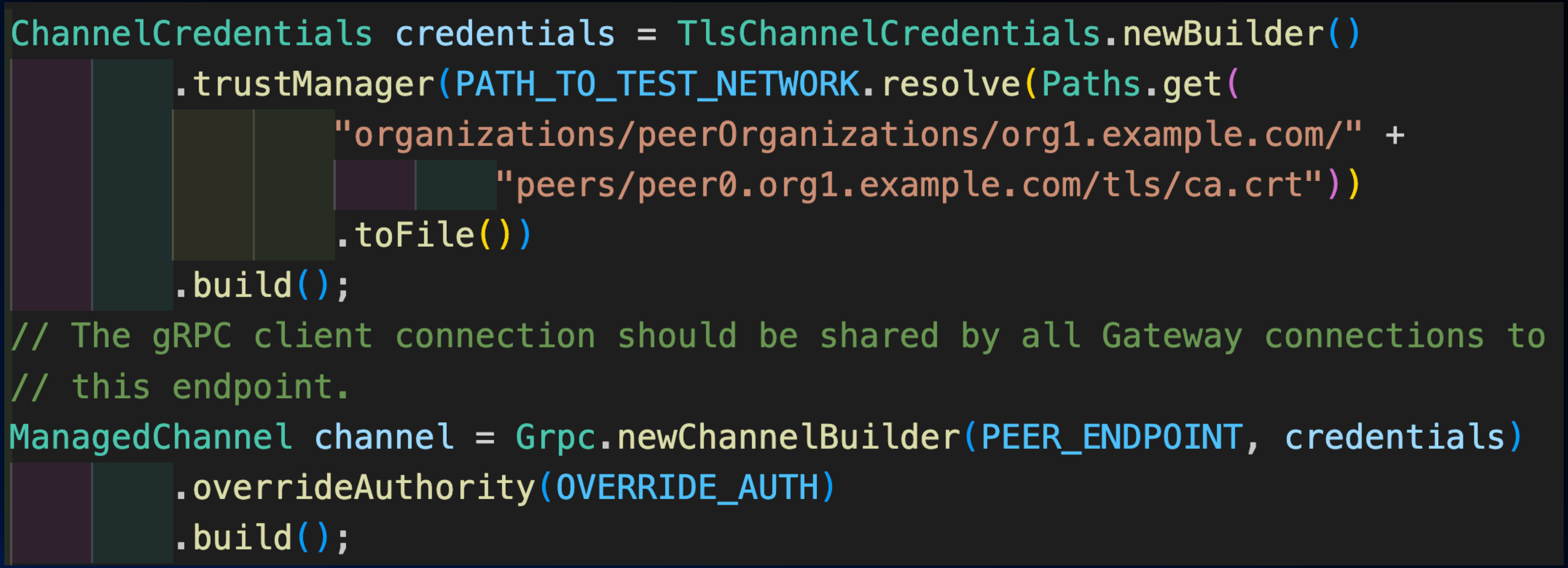
- Install dependencies
- At least one reachable peer endpoint
- Crypto things
    - Certificate of the peer’s CA
    - Certificate of the interacting user
    - Private key of the interacting user
    - You can find them in the `organizations` directory inside `test-network` or by using a wallet

Client Application - Steps

1. Setup the connection to the channel via grpc
2. Setup the identity to perform the operations on the channel
3. Setup the signer of such operations
4. Establish the connection to the Fabric Gateway service (peer)
5. Get a reference of the chaincode
6. Interact with it

- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Setup the connection

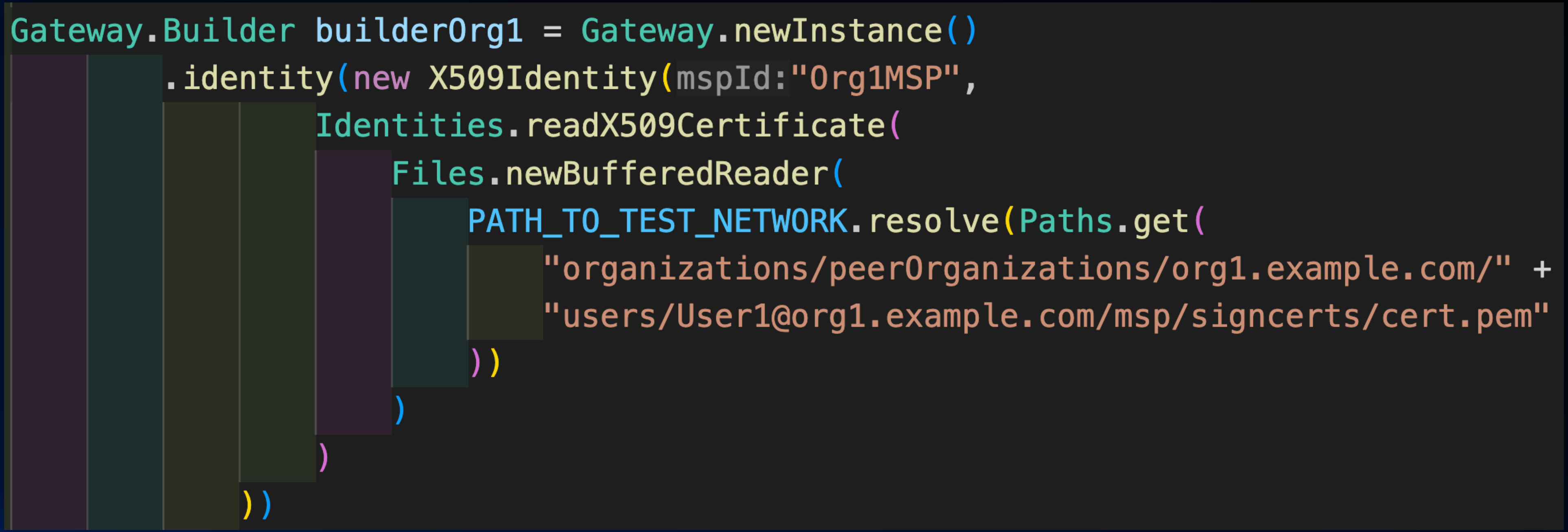


```java
ChannelCredentials credentials = TlsChannelCredentials.newBuilder()
        .trustManager(PATH_TO_TEST_NETWORK.resolve(Paths.get(
                "organizations/peerOrganizations/org1.example.com/" +
                        "peers/peer0.org1.example.com/tls/ca.crt"))
                .toFile())
        .build();
// The gRPC client connection should be shared by all Gateway connections to
// this endpoint.
ManagedChannel channel = Grpc.newChannelBuilder(PEER_ENDPOINT, credentials)
        .overrideAuthority(OVERRIDE_AUTH)
        .build();
```

1. Use the certificate of the peer's CA
2. Use the peer endpoint (`localhost:7051`)
3. Use the container name of the peer
4. Initiate the connection to the channel

- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Setup the identity



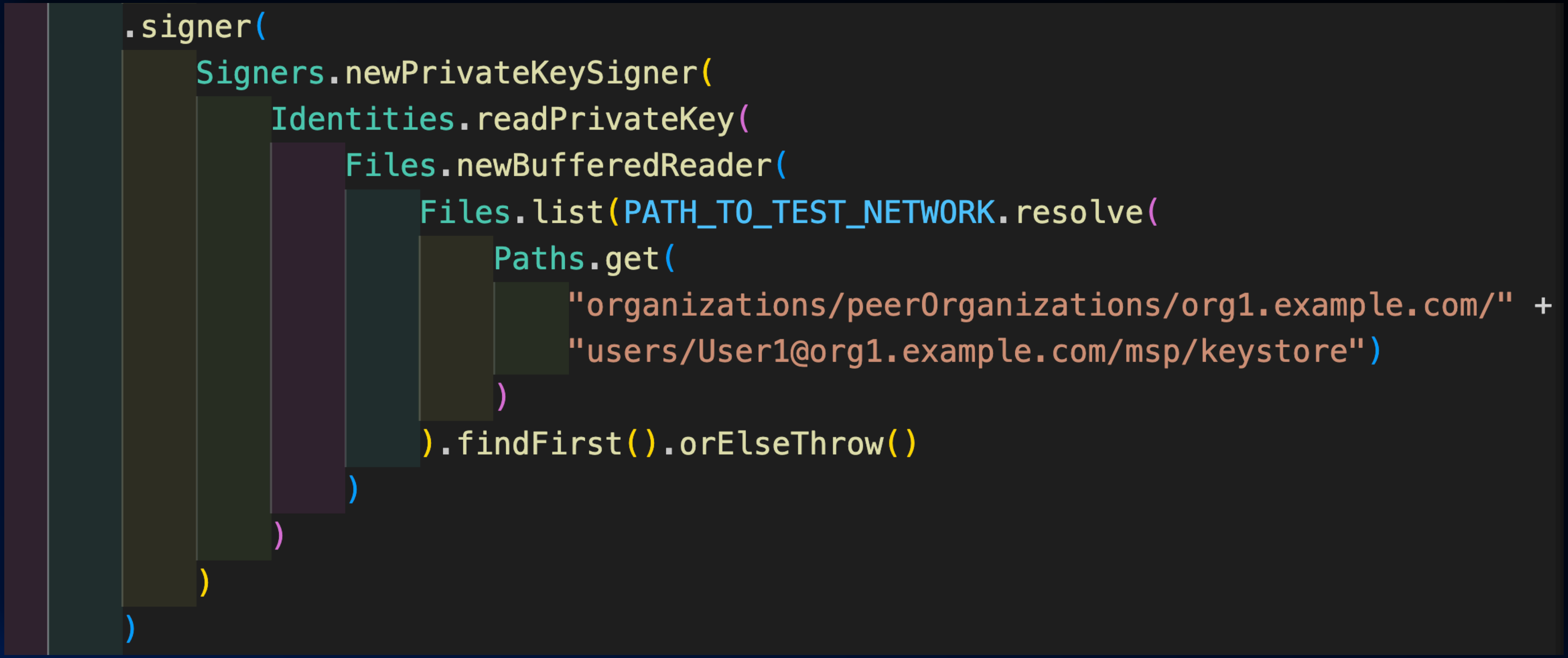
```
Gateway.Builder builderOrg1 = Gateway.newInstance()
        .identity(new X509Identity(mspId:"Org1MSP",
                Identities.readX509Certificate(
                    Files.newBufferedReader(
                        PATH_TO_TEST_NETWORK.resolve(Paths.get(
                            "organizations/peerOrganizations/org1.example.com/" +
                            "users/User1@org1.example.com/msp/signcerts/cert.pem"
                        ))
                    )
                )
            ))
```

1. Define the MSP
2. Get the certificate of the user
3. Pass these information to `.identity()` of the Gateway builder

- The identity is the entity that interacts with the chaincode

- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Setup the signer



```
.signer(
    Signers.newPrivateKeySigner(
        Identities.readPrivateKey(
            Files.newBufferedReader(
                Files.list(PATH_TO_TEST_NETWORK.resolve(
                    Paths.get(
                        "organizations/peerOrganizations/org1.example.com/" +
                        "users/User1@org1.example.com/msp/keystore")
                    )
                ).findFirst().orElseThrow()
            )
        )
    )
)
```

1. Get the user's private key
2. Pass these information to `.signer()` of the Gateway builder

- The signer is the one signing messages sent to the channel

- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Establish the connection

1. Pass the grpc channel connection to the Gateway builder

```
.connection(channel)
```

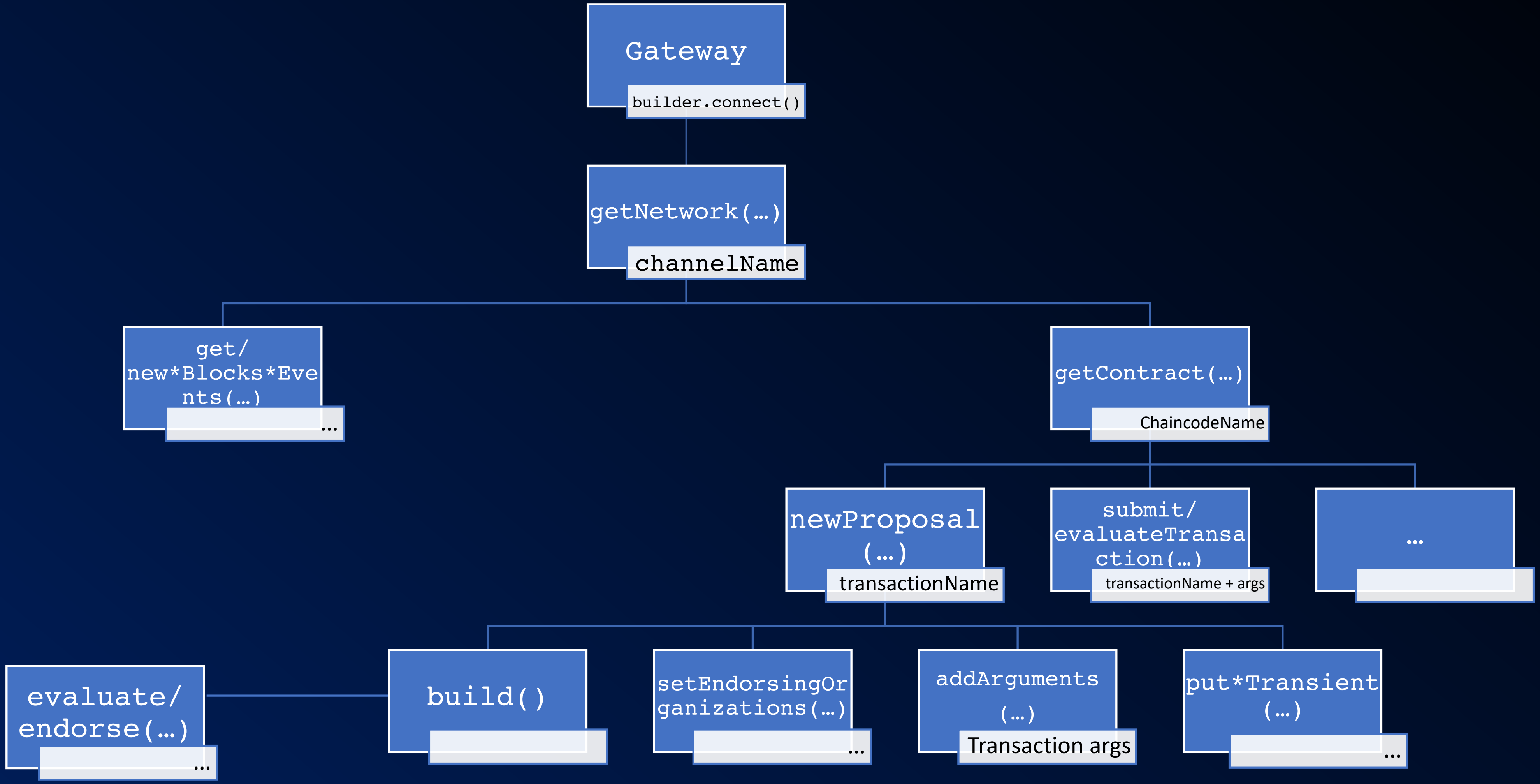
2. Call `.connect()` of the Gateway builder

```
try (Gateway gatewayOrg1 = builderOrg1.connect()) {
```

- Now you have a Gateway object connected to the channel

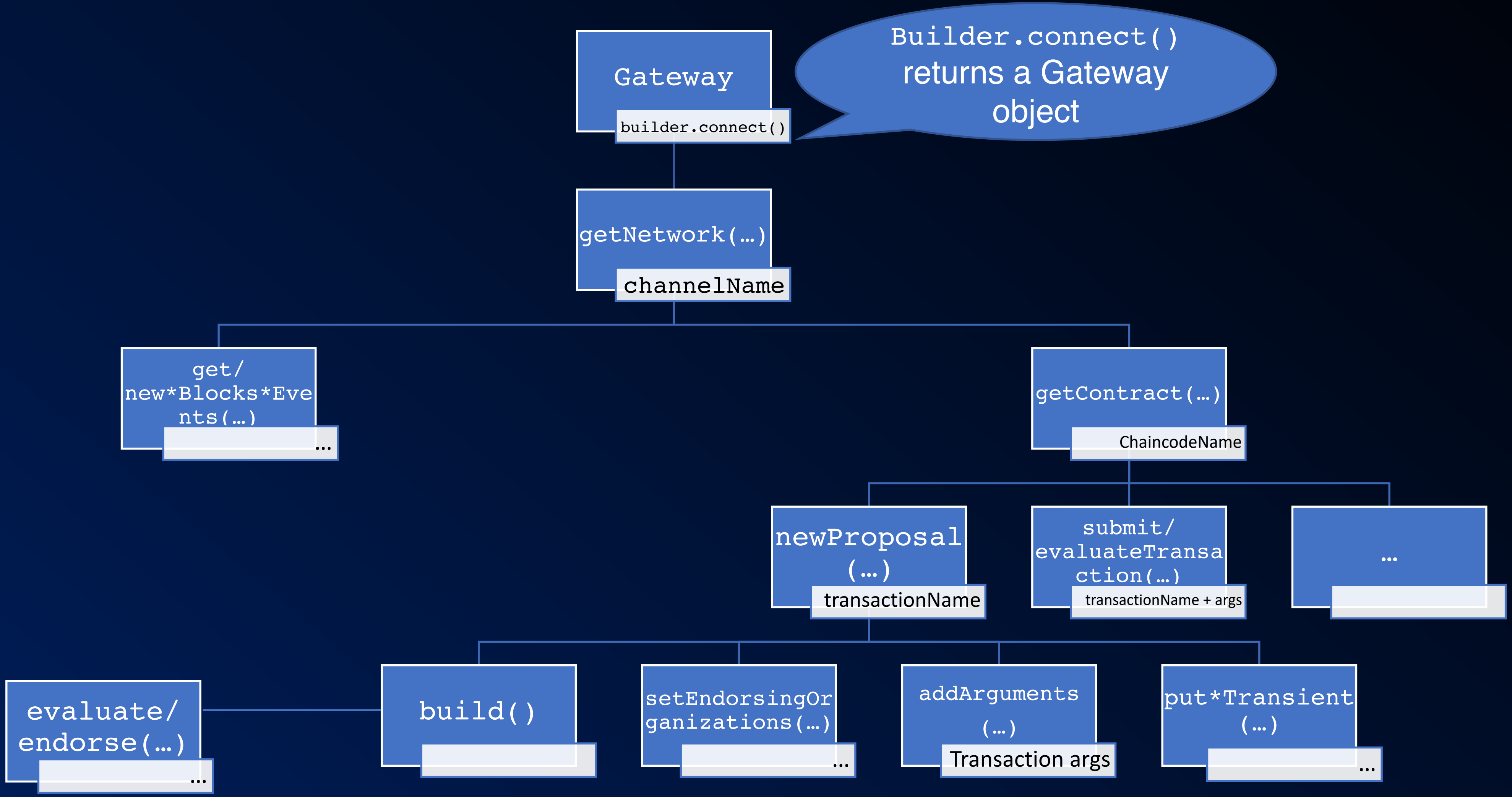
- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Things to interact with the channel



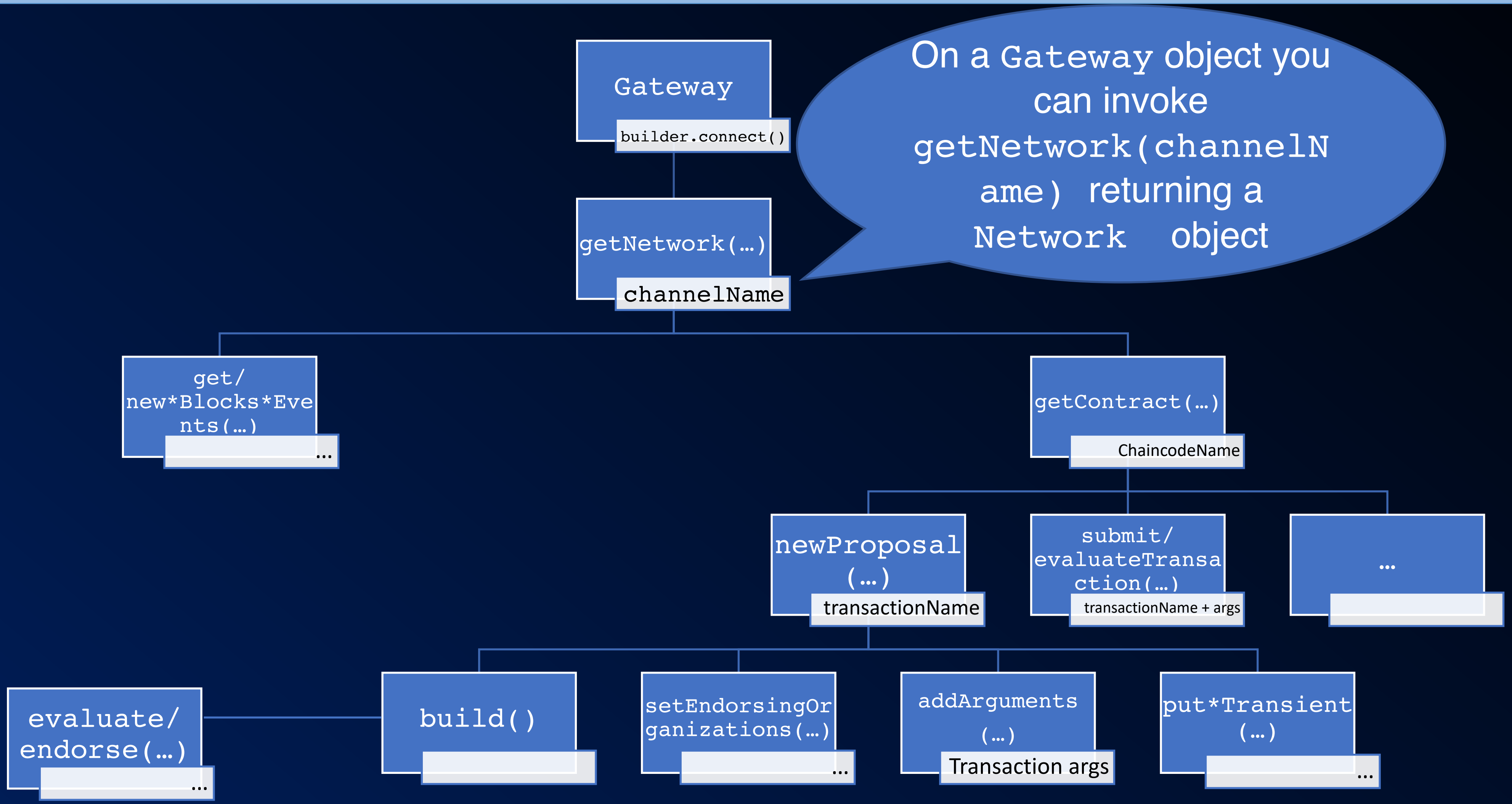
- **Documentation:**
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Things to interact with the channel



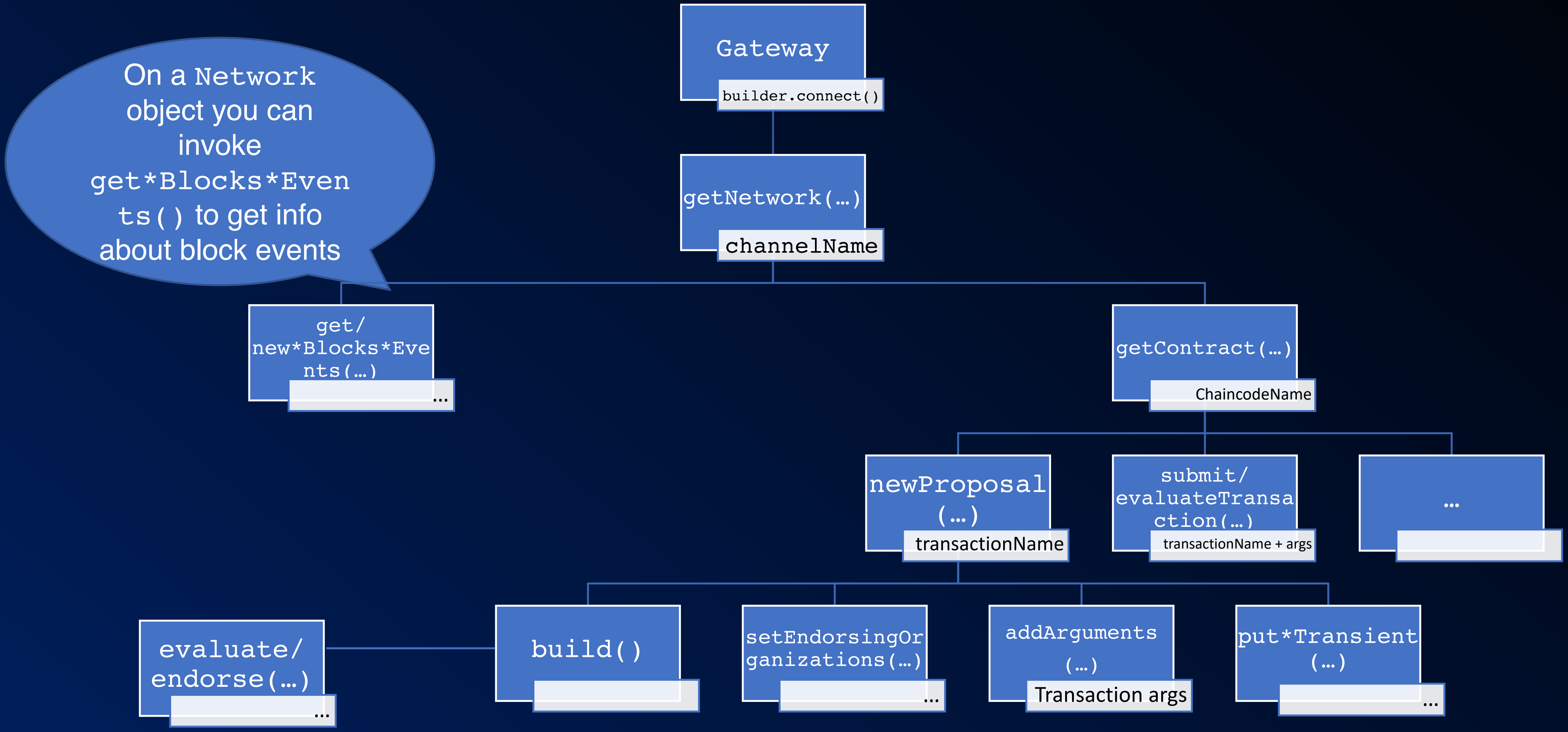
- **Documentation:**
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Things to interact with the channel



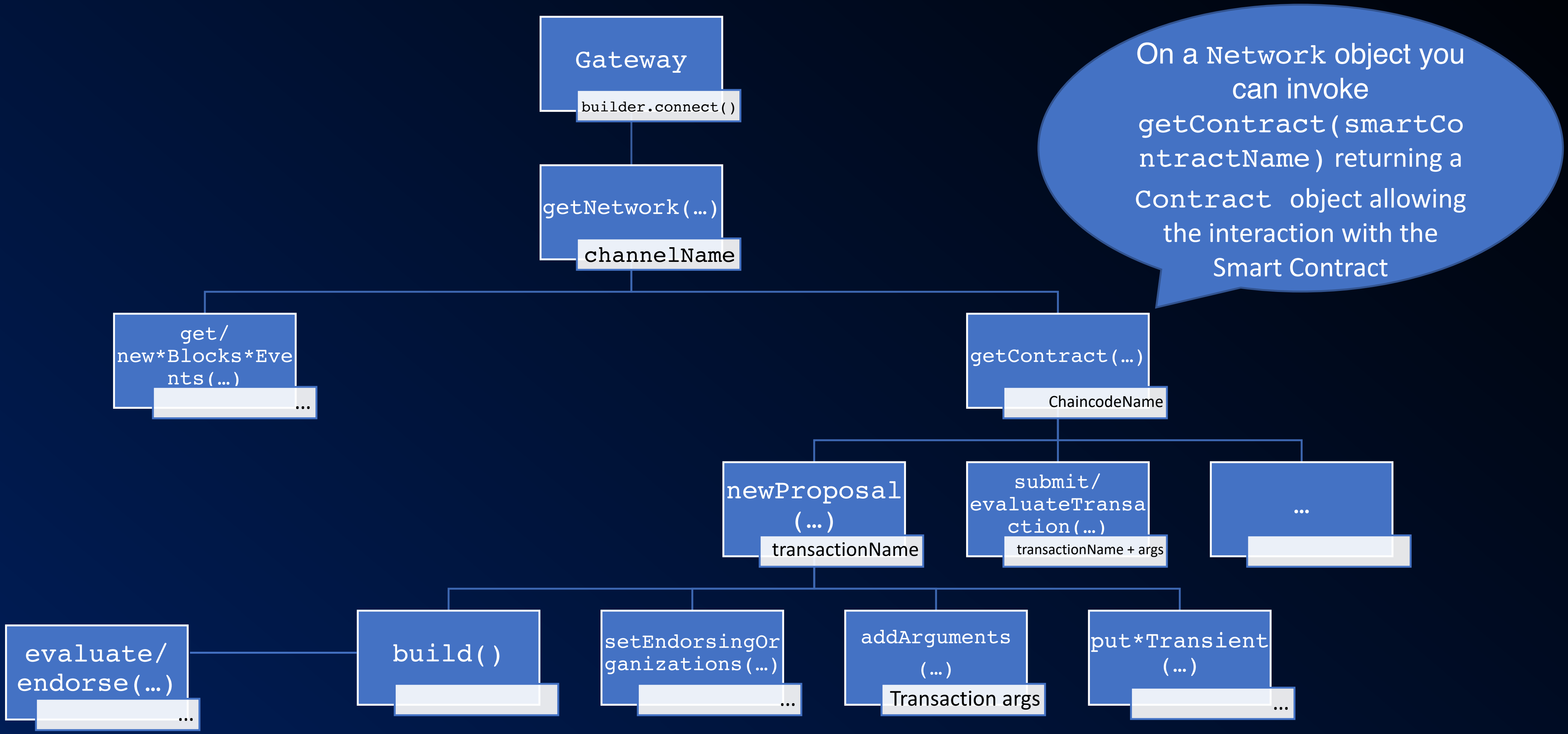
- **Documentation:**
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Things to interact with the channel



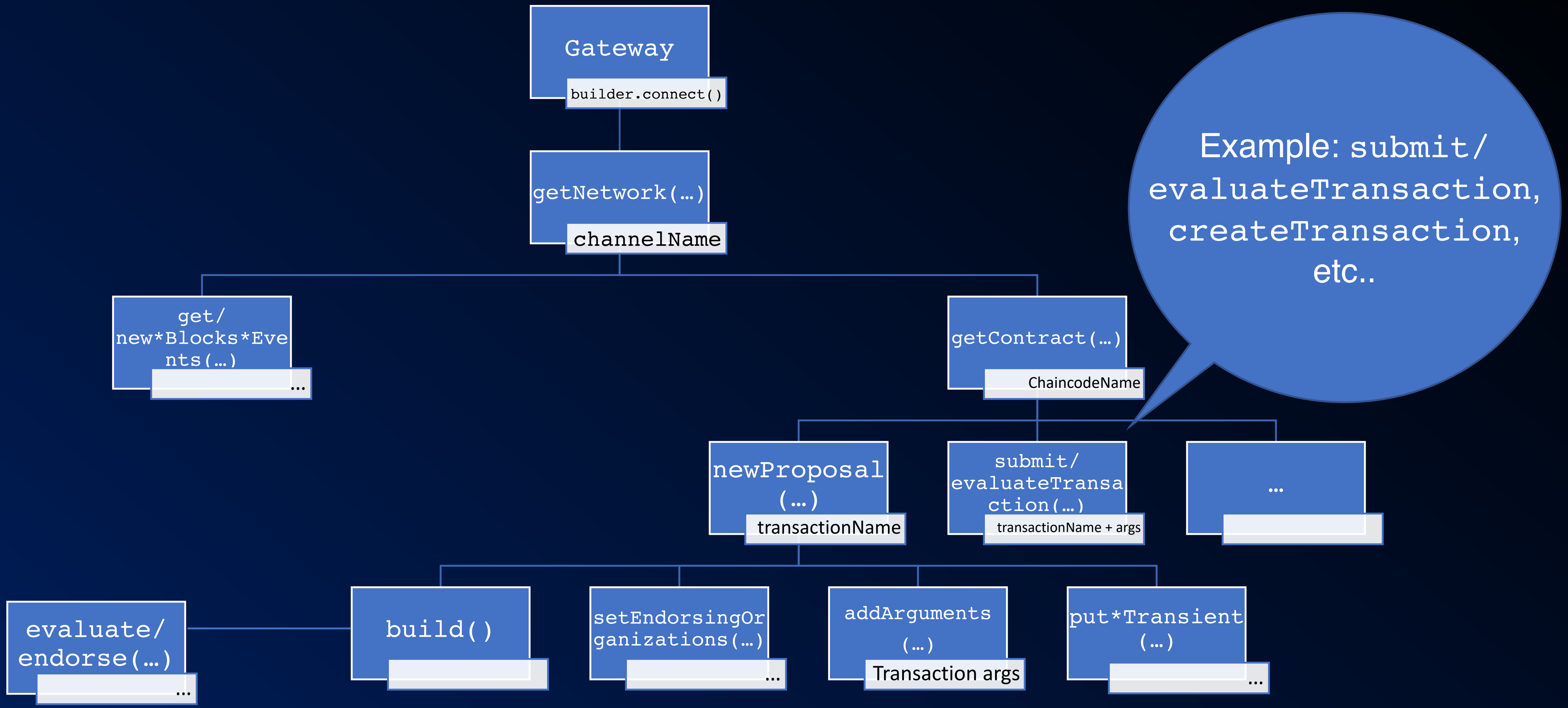
- **Documentation:**
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Things to interact with the channel



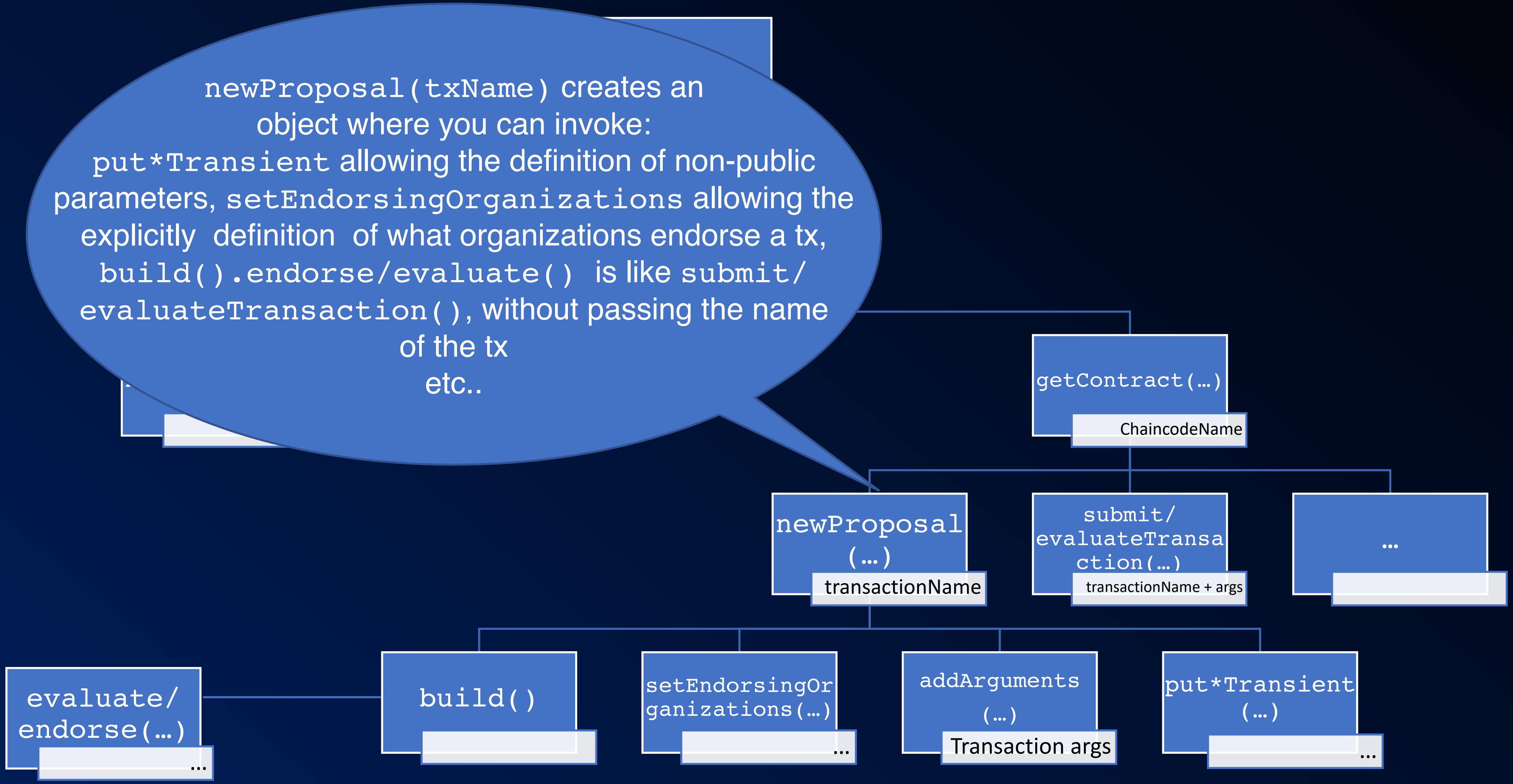
- **Documentation:**
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Things to interact with the channel



- **Documentation:**
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

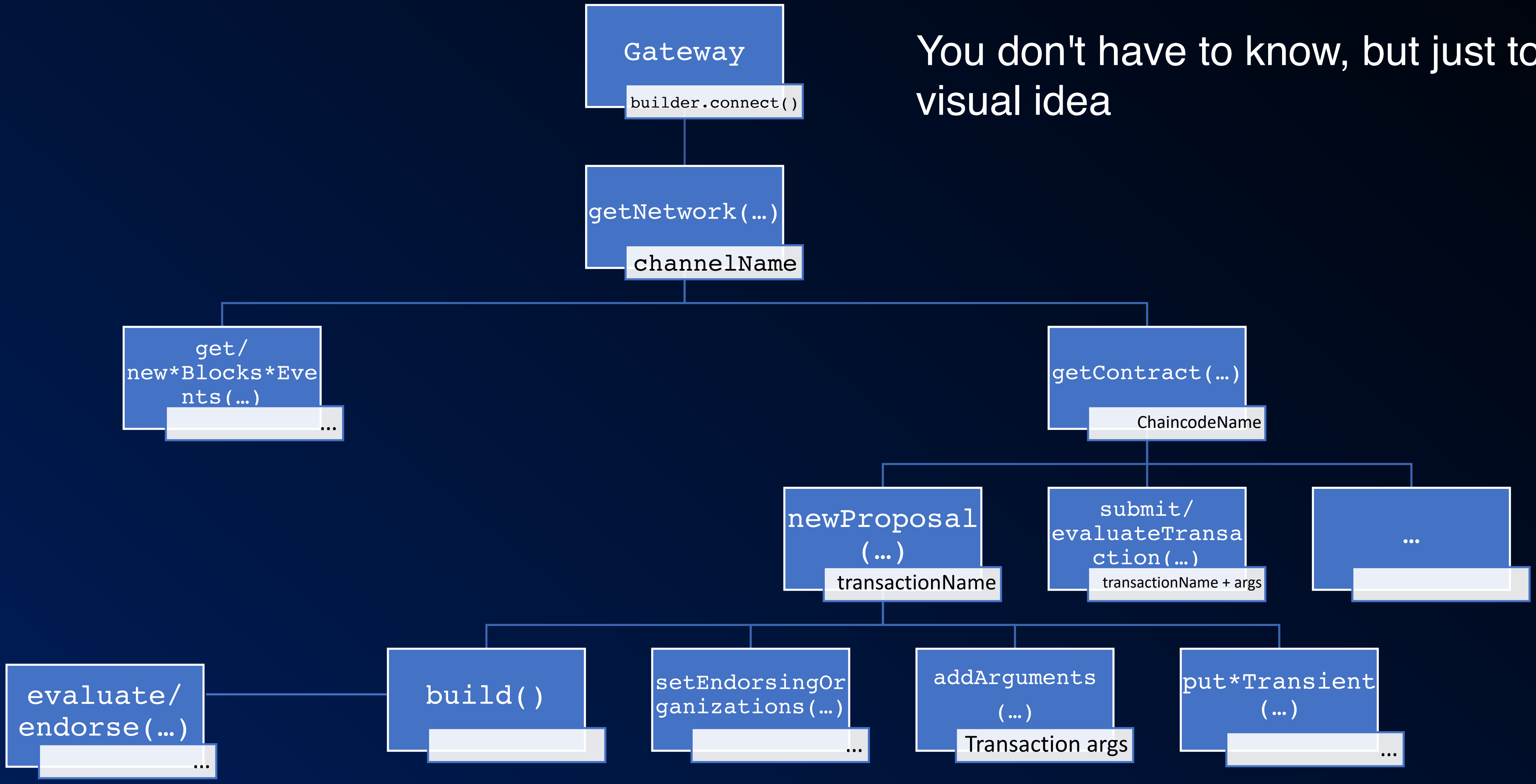
Client Application - Things to interact with the channel


newProposal(txName) creates an object where you can invoke: put*Transient allowing the definition of non-public parameters, setEndorsingOrganizations allowing the explicitly definition of what organizations endorse a tx, build().endorse/evaluate() is like submit/evaluateTransaction(), without passing the name of the tx etc..
getContract(…)
ChaincodeName
newProposal(…)
transactionName
submit/evaluateTransaction(…)
transactionName + args
…
evaluate/endorse(…)
…
build()
setEndorsingOrganizations(…)
…
addArguments(…)
Transaction args
put*Transient(…)
…

- **Documentation:**
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

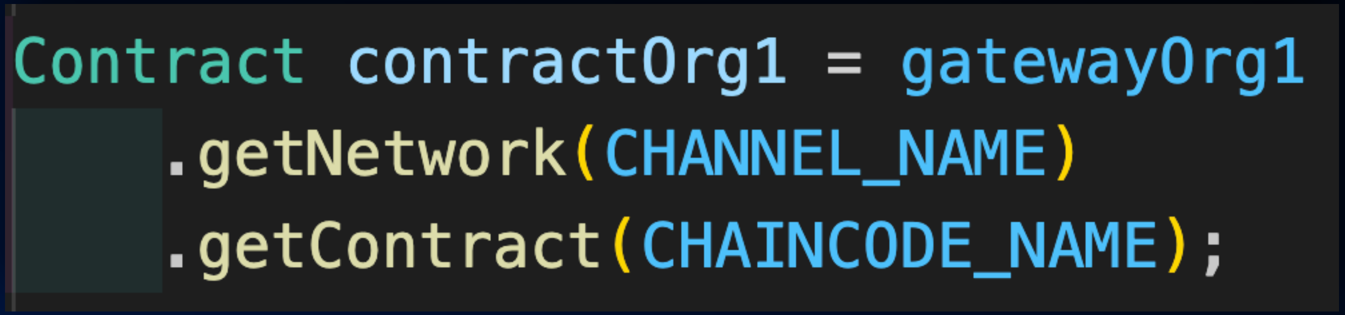
Client Application - Things to interact with the channel

You don't have to know, but just to get a visual idea



- # Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Get the Chaincode reference



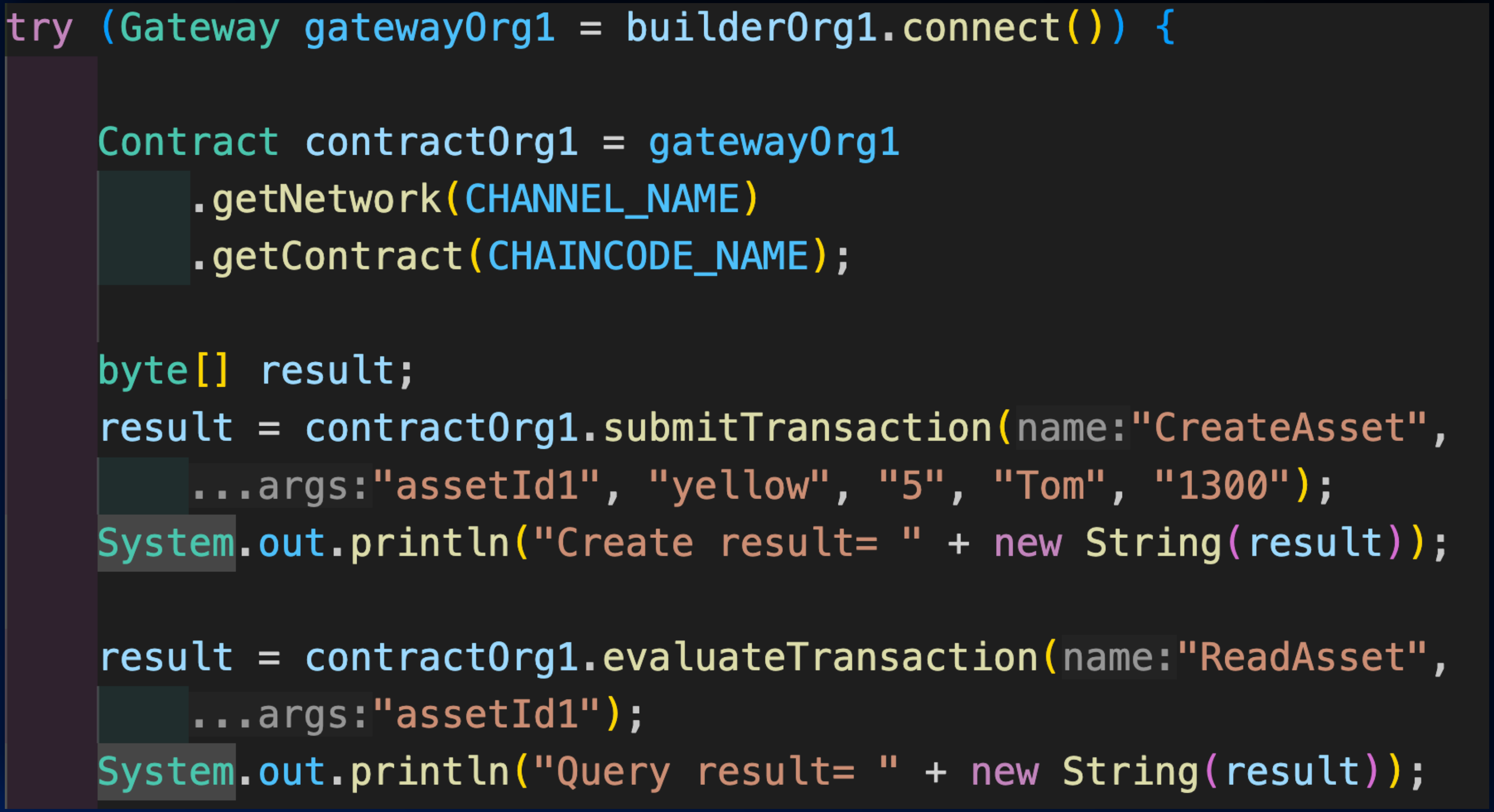
```
Contract contractOrg1 = gatewayOrg1
        .getNetwork(CHANNEL_NAME)
        .getContract(CHAINCODE_NAME);
```

1. Get the Network reference by passing the channel name
2. Get the Smart Contract/Chaincode reference by passing the chaincode name

- Now you can send Transactions

- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

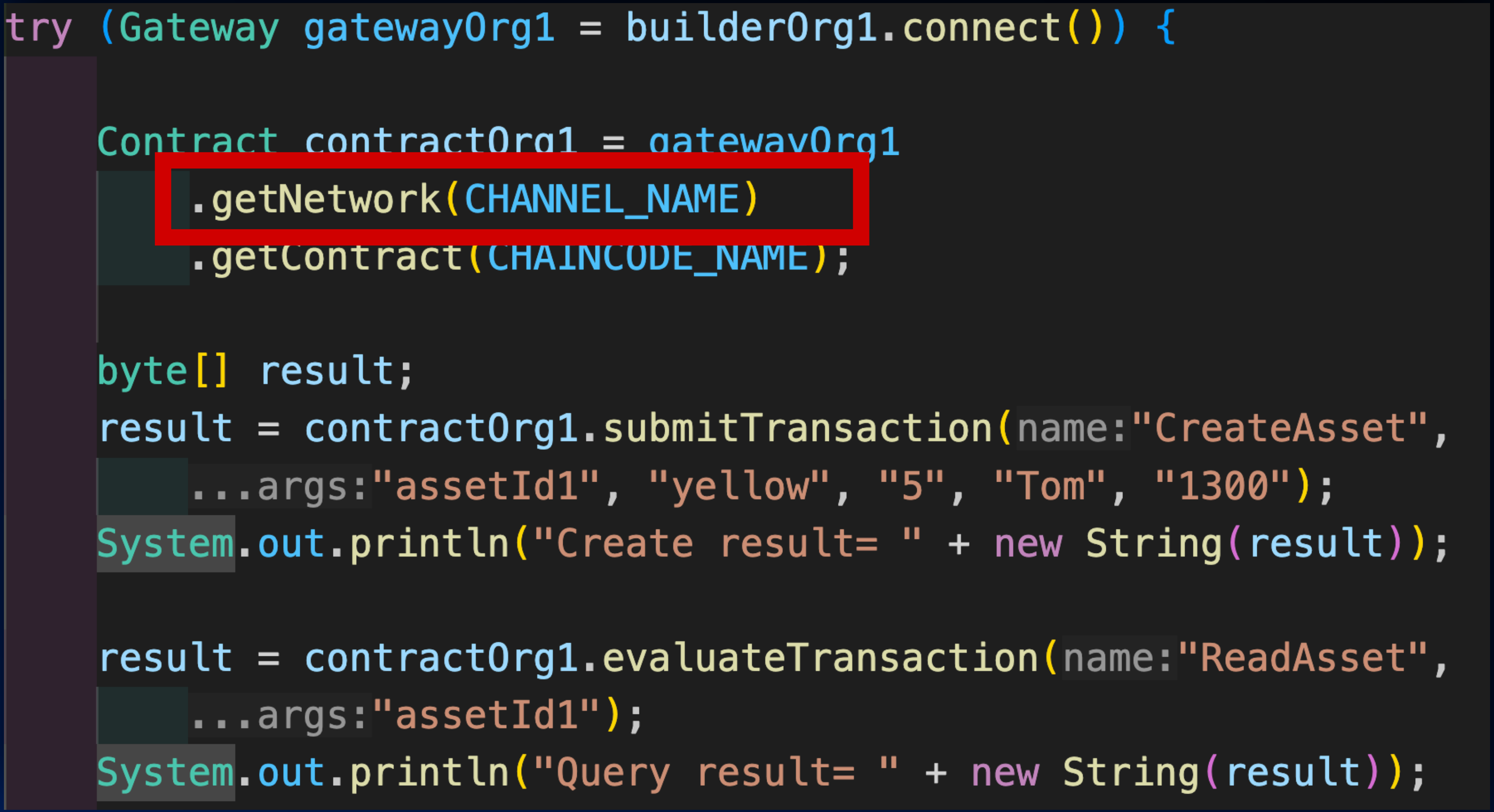
Client Application - Send Transactions



```
try (Gateway gatewayOrg1 = builderOrg1.connect()) {

    Contract contractOrg1 = gatewayOrg1
        .getNetwork(CHANNEL_NAME)
        .getContract(CHAINCODE_NAME);

    byte[] result;
    result = contractOrg1.submitTransaction(name:"CreateAsset",
        ...args:"assetId1", "yellow", "5", "Tom", "1300");
    System.out.println("Create result= " + new String(result));

    result = contractOrg1.evaluateTransaction(name:"ReadAsset",
        ...args:"assetId1");
    System.out.println("Query result= " + new String(result));
```

- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

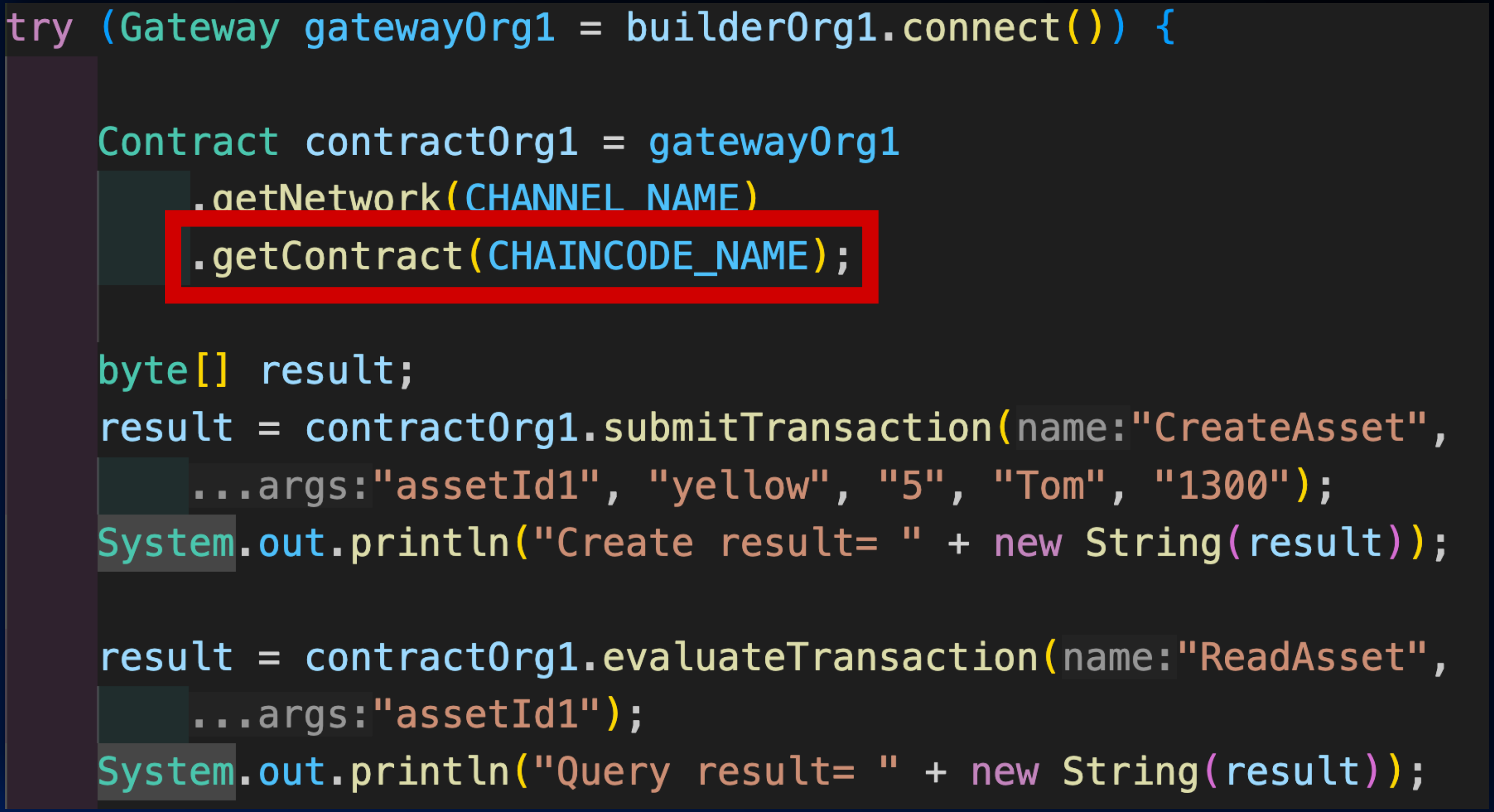
Client Application - Send Transactions



```
try (Gateway gatewayOrg1 = builderOrg1.connect()) {

    Contract contractOrg1 = gatewayOrg1
            .getNetwork(CHANNEL_NAME)
            .getContract(CHAINCODE_NAME);

    byte[] result;
    result = contractOrg1.submitTransaction(name:"CreateAsset",
            ...args:"assetId1", "yellow", "5", "Tom", "1300");
    System.out.println("Create result= " + new String(result));

    result = contractOrg1.evaluateTransaction(name:"ReadAsset",
            ...args:"assetId1");
    System.out.println("Query result= " + new String(result));
```

- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

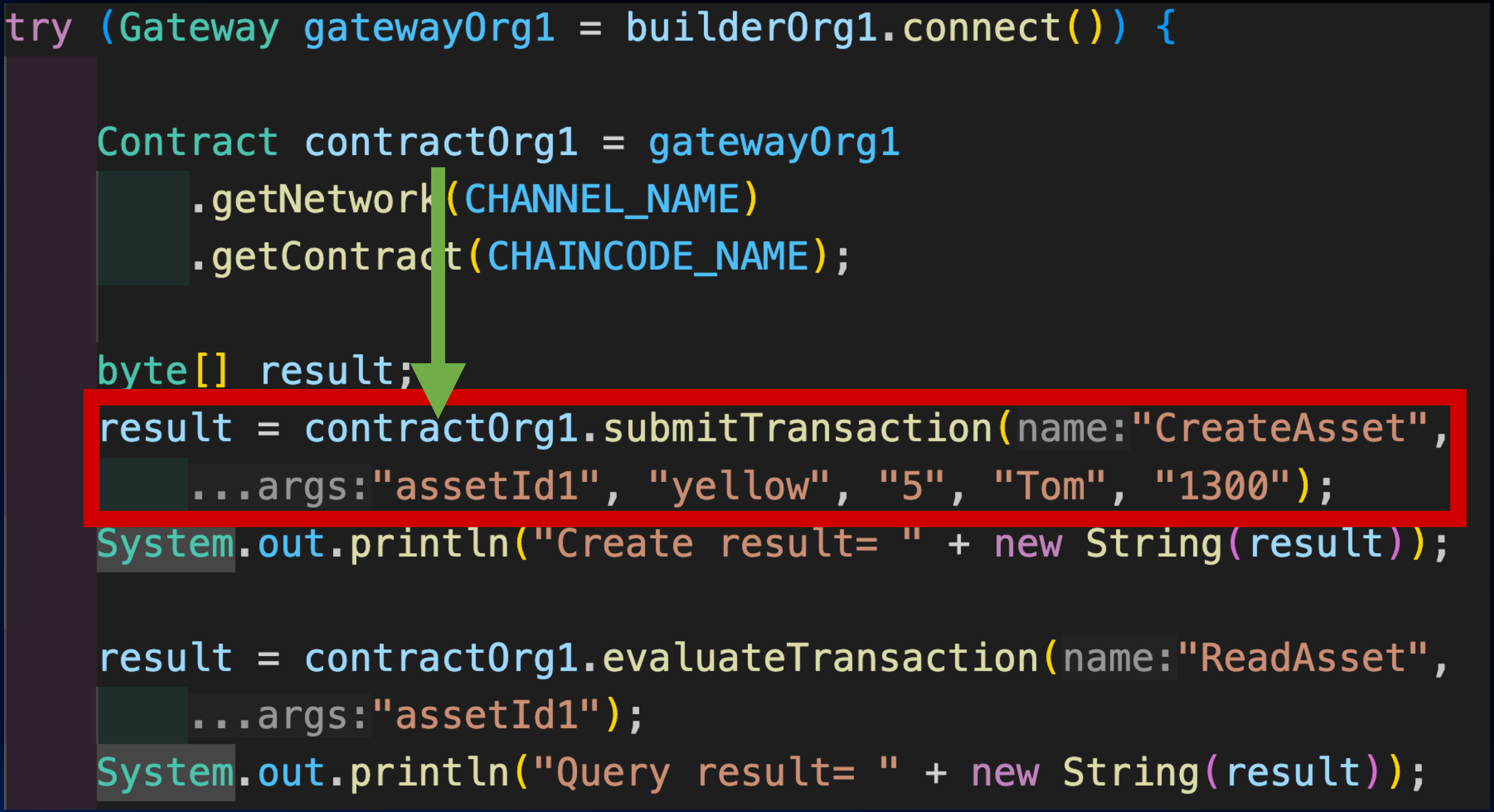
Client Application - Send Transactions



```
try (Gateway gatewayOrg1 = builderOrg1.connect()) {

    Contract contractOrg1 = gatewayOrg1
        .getNetwork(CHANNEL_NAME)
        .getContract(CHAINCODE_NAME);

    byte[] result;
    result = contractOrg1.submitTransaction(name:"CreateAsset",
        ...args:"assetId1", "yellow", "5", "Tom", "1300");
    System.out.println("Create result= " + new String(result));

    result = contractOrg1.evaluateTransaction(name:"ReadAsset",
        ...args:"assetId1");
    System.out.println("Query result= " + new String(result));
```

- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

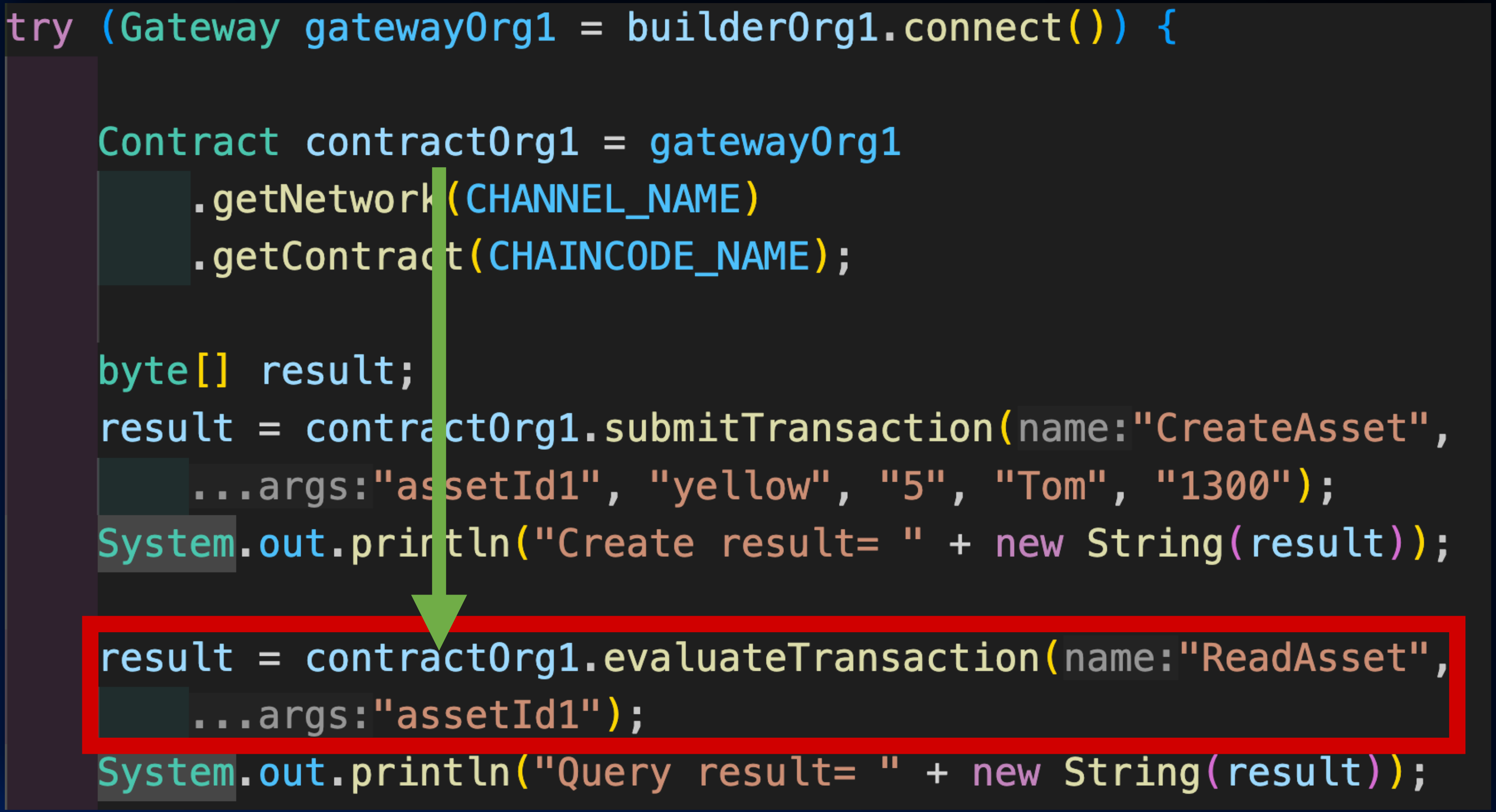
Client Application - Send Transactions



```
try (Gateway gatewayOrg1 = builderOrg1.connect()) {

    Contract contractOrg1 = gatewayOrg1
            .getNetwork(CHANNEL_NAME)
            .getContract(CHAINCODE_NAME);

    byte[] result;
    result = contractOrg1.submitTransaction(name:"CreateAsset",
            ...args:"assetId1", "yellow", "5", "Tom", "1300");
    System.out.println("Create result= " + new String(result));

    result = contractOrg1.evaluateTransaction(name:"ReadAsset",
            ...args:"assetId1");
    System.out.println("Query result= " + new String(result));
```

- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

Client Application - Send Transactions



```
try (Gateway gatewayOrg1 = builderOrg1.connect()) {

    Contract contractOrg1 = gatewayOrg1
        .getNetwork(CHANNEL_NAME)
        .getContract(CHAINCODE_NAME);

    byte[] result;
    result = contractOrg1.submitTransaction(name:"CreateAsset",
        ...args:"assetId1", "yellow", "5", "Tom", "1300");
    System.out.println("Create result= " + new String(result));

    result = contractOrg1.evaluateTransaction(name:"ReadAsset",
        ...args:"assetId1");
    System.out.println("Query result= " + new String(result));
```

- Documentation:
- (Modified version of) https://github.com/hyperledger/fabric-samples/tree/main/asset-transfer-basic/application-gateway-java
- https://hyperledger-fabric.readthedocs.io/en/release-2.5/write_first_app.html
- https://hyperledger.github.io/fabric-gateway/main/api/java/ and internals

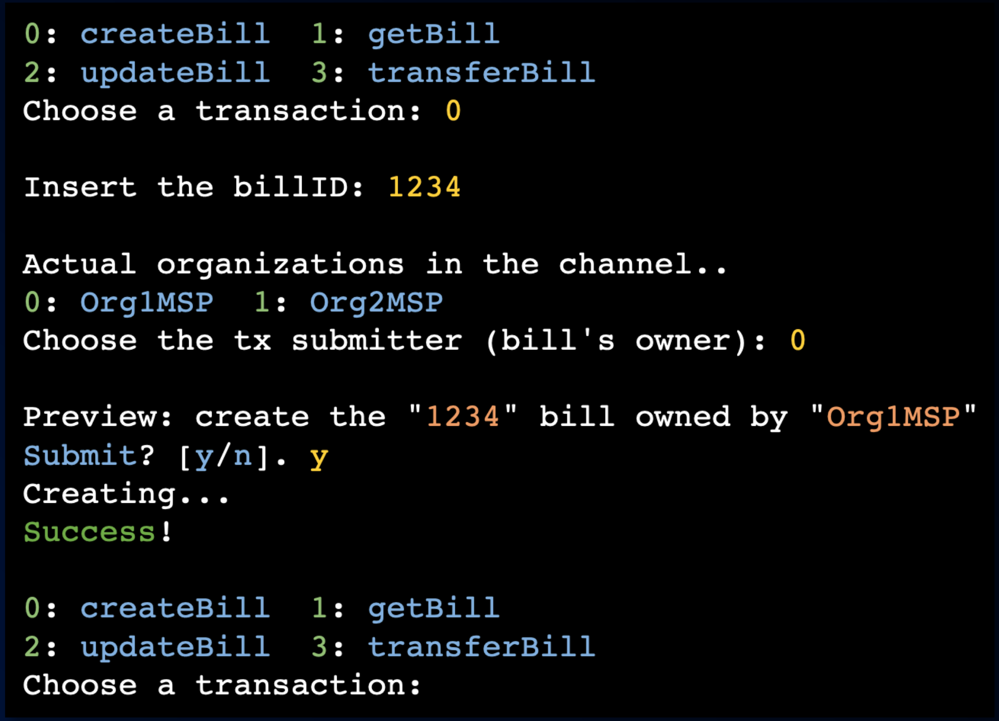
Client Application - Hands on

1. Download the application template project from https://github.com/avolast/hyperledger-fabric-simple-template
2. Import the gradle project in an IDE
3. Implement a simple client application for the interaction with the `CarContract`.
    - Pay attention to the `PATH_TO_TEST_NETWORK` field!
4. Execute `./gradlew run` or run the project directly in the IDE
5. Exercise to solve
6. Next lesson: Introduction to Kubernetes

Client Application - Exercise

Document ownership transfer (continue)

- Create a client application allowing via a multiple choice menu the usage of the Smart Contract of the exercise done last time
    - For each menu choice, ask for appropriate input
    - Each tx request must be submitted as Org1 or Org2
- Who wants, develop a simple GUI (at your convenience).
- Use the language you prefer among those usable.
- An example of menu and interaction is reported in figure
    - Obviously your implementation doesn't have to be the same (especially for the colors) …



```
0: createBill   1: getBill
2: updateBill   3: transferBill
Choose a transaction: 0

Insert the billID: 1234

Actual organizations in the channel..
0: Org1MSP   1: Org2MSP
Choose the tx submitter (bill's owner): 0

Preview: create the "1234" bill owned by "Org1MSP"
Submit? [y/n]. y
Creating...
Success!

0: createBill   1: getBill
2: updateBill   3: transferBill
Choose a transaction:
```

- Hint: connect the application to the channel using 2 gateways: one for Org1 and the other for Org2. Once you have chosen the tx submitter, use one of the two gateways appropriately.