```
package main

import
(
    "encoding/json"
    "fmt"
    "log"
    "github.com/hyperledger/fabric-contract-api-go/contractapi"
)

// AssetTransfer contract for managing assets
type AssetTransfer struct
{
    contractapi.Contract
}

// Asset represents an asset with basic attributes
type Asset struct
{
    ID            string `json:"ID"`
    Owner         string `json:"Owner"`
    Color         string `json:"Color"`
    Size          int    `json:"Size"`
    AppraisedValue int    `json:"AppraisedValue"`
}
```

// CreateAsset initializes a new asset

```
func (t *AssetTransfer) CreateAsset(ctx
contractapi.TransactionContextInterface, id string, owner string, color
string, size int, appraisedValue int) error
 {
    asset := Asset

    {

        ID:             id,

        Owner:          owner,

        Color:          color,

        Size:           size,

        AppraisedValue: appraisedValue,

    }
    assetJSON, err := json.Marshal(asset)

    if err != nil

    {

        return err

    }
    return ctx.GetStub().PutState(id, assetJSON)

}


// ReadAsset retrieves an asset by its ID

func (t *AssetTransfer) ReadAsset(ctx
contractapi.TransactionContextInterface, id string) (*Asset, error) {

    assetJSON, err := ctx.GetStub().GetState(id)

    if err != nil
    {

        return nil, fmt.Errorf("failed to read from world state: %v", err)
    }

    if assetJSON == nil

     {

        return nil, fmt.Errorf("asset %s does not exist", id)

    }
    var asset Asset

    err = json.Unmarshal(assetJSON, &asset)

    if err != nil

     {

        return nil, err

    }
    return &asset, nil

}

// UpdateAsset modifies an existing asset

func (t *AssetTransfer) UpdateAsset(ctx
contractapi.TransactionContextInterface, id string, owner string, color
string, size int, appraisedValue int) error
{

    asset, err := t.ReadAsset(ctx, id)

    if err != nil

     {

        return err

    }
    asset.Owner = owner
    asset.Color = color

    asset.Size = size

    asset.AppraisedValue = appraisedValue
    assetJSON, err := json.Marshal(asset)

    if err != nil {

        return err

    }
    return ctx.GetStub().PutState(id, assetJSON)

}

// DeleteAsset removes an asset by its ID

func (t *AssetTransfer) DeleteAsset(ctx
contractapi.TransactionContextInterface, id string) error {

    return ctx.GetStub().DelState(id)

}


// GetAllAssets retrieves all assets

func (t *AssetTransfer) GetAllAssets(ctx
contractapi.TransactionContextInterface) ([]*Asset, error) {

    queryString := `{"selector": {}}`

    resultsIterator, err := ctx.GetStub().GetQueryResult(queryString)

    if err != nil

     {

    return nil, err


    }

    defer resultsIterator.Close()
    var assets []*Asset

    for resultsIterator.HasNext()
```

```
     {
       queryResponse, err := resultsIterator.Next()
       if err != nil {
           return nil, err
       }
       var asset Asset
       err = json.Unmarshal(queryResponse.Value, &asset)
       if err != nil

       {
           return nil, err
       }
       assets = append(assets, &asset)
    }
    return assets, nil
}
func main() {
    chaincode, err := contractapi.NewChaincode(new(AssetTransfer))
    if err != nil {

        log.Panicf("Error creating asset-transfer chaincode: %v", err)

    }
    if err := chaincode.Start(); err != nil {

        log.Panicf("Error starting asset-transfer chaincode: %v", err)

    }

}
```