```
struct factoryPayment {
 int sum;
 int monthsLeft;
};
```

```
struct moneyLoan {
 int sum;
 int monthsLeft;
};
```

```
struct playerProperty {
 int commonFactories;
 int autoFactories;
 int balance;
 int raw;
 int production;
 vector<factoryPayment> factoryLoans;
 vector<moneyLoan> moneyLoans;
};
```

```
struct playerInfo {
 int commonFactories;
 int autoFactories;
 int balance;
 int raw;
 int production;
};
```

```
typedef vector<playerInfo> infoTable;
```

```
class Bank {
 vector<playerProperty> properties;
 vector<infoTable> infoTables;
 ...
 bid makeBids();
 ...
};
```

```
struct happyCaseMove {
 int rawGift;
 int prodGift;
 int moneyGift;
};
```

```
struct request {
 int count;
 int cost
};
```

```
struct bid {
 request rawSellBid;
 request prodBuyBid;
 happyCaseOccasion case;
};
```

```
struct happyCaseOccasion {
 int index;
 int target;
};
```

```
struct playerMove {
 request rawBuyRequest;
 request prodSellRequest;

 int factoriesToAuto;
 int buyFactories;
 int toProductionCommon;
 int toProductionAuto;
 int loan;

 happyCaseMove gift;
};
```

```
class playerAlgorithm {
 virtual playerMove getMove(playerPropetry, vector<infoTable>, bid) = 0;

 virtual string getCode() = 0;
};
```

```
class myAlgorithm: playerAlgorithm {
 playerMove getMove(playerProperty, vector<infoTable>, bid) override {...}

 string getCode() override {...}
};
```

```
class Interface {
 ...
};
```

```
struct playerEntity {
 enum playerType {comp, human} type;
 playerAlgorithm* algorithm;
};
```

```
class playerManager {
 vector<playerEntity> players;
 ...
};
```