# Contents



# 1 Use Case Diagrams

**Use Case Diagrams**

Use Case diagrams

- are not showing structure of system,
- are not showing behavior of system,
- shows interaction between system and outer participants (people, other systems).
- Use Case diagrams helps analytic to communicate with customer.
  - What roles users have?
  - How will users use system?

**What Are Use Case Diagrams?**

- Use Cases describe system usage scenarios.
- They describe user roles.
- They describe system from users point of view.

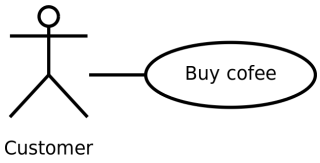
**Example**

Example: We want to design cofee machine.

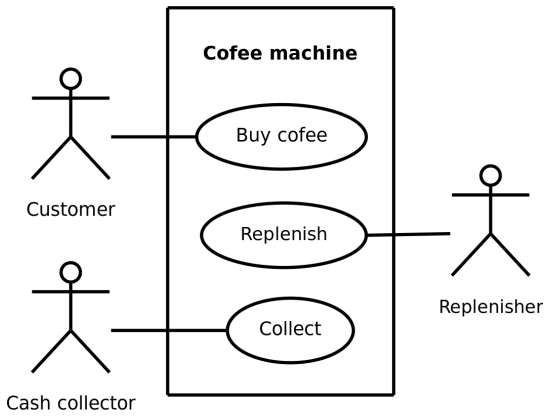
1. We talk with potential users about possible usage of the machine.
   - Most users sais that they want to buy the cofee ;-) – we have scenario = Use Case (č. případ užití)
2. We identify all participants (actors) that plays its role in the scenario.
   - In this use case is the only actor the customer that wants to buy cofee.

3. How the scenario looks like? What are preconditions of the scenario usage? What are scenario consequences?
4. What other scenarios for the participant are possible?
5. How to modify scenario when customer realises that he/she has not enough money?
6. Are there other participants that interact with cofee machine?

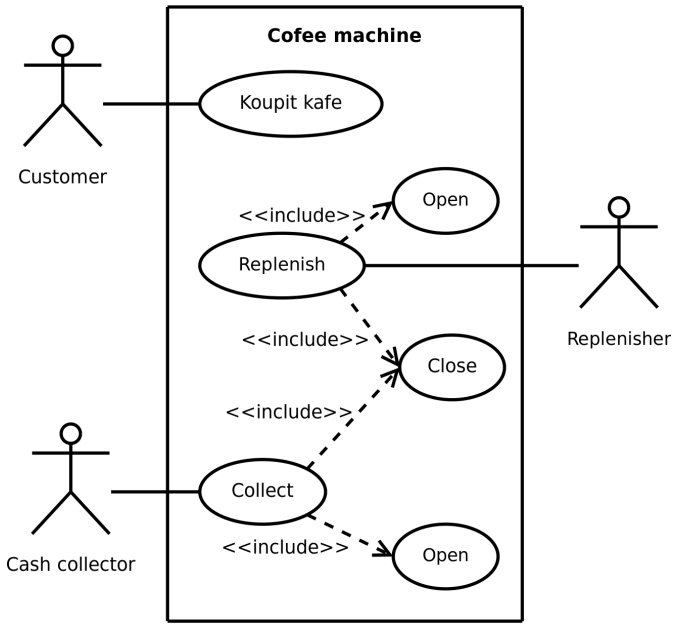
**Use Cases in UML – I.**



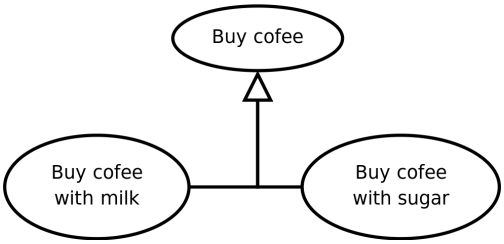

**Use Cases in UML – II.**



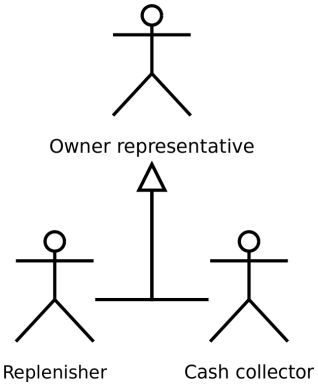

**Use Cases in UML – III.**




**Use Cases in UML – IV.**



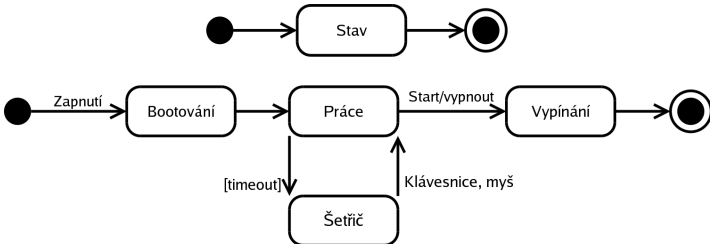

**Use Cases in UML – V.**



# 2 State Diagrams

### State Diagrams

- Object is offten in few distinct states during its' life.
  - Creation, work (multiple states), destruction, ...
- State diagram describes all states and *transitions* between them.




# 3 What has not been (but should be) said

### Procedure Fail

- It can be used only in constructor.
- It means that there is some error during constructor invokation and object creation should be undone.
- The New function (or procedure) will release object in heap.
- The pointer to object is set to *nil* and object is unusable.

### Inherited

- With keyword *Inherited* is invoked method from ancestor.
- It is used in inherited methods, constructors and destructors

```
function CAncestor.Method(var1:Integer):string;
begin
  ...
end;
function CDescendant.Method(var:Integer):string;
var temp:string;
begin
  temp:=Inherited Method(var);
  ...
end;
```

**Examples of Inherited**

```
constructor CDescendant.Init;
begin
  inherited Init;
  ... further constructors' code
end;


destructor CDescendant.Done;
begin
  ... destructors' code
  inherited Done;
end;
```

# 4 Some Notes to Semester Work

**Pár poznámek k semestrální práci**

- Minimální použití vícevrstvého modelu aplikace je oddělení datové vrstvy.
  - Mezi datovou a business vrstvou probíhá komunikace typu "Vytvoř nového zákazníka" nebo "Dej na fakturu ... položku ...".
  - Není to komunikace typu "Do souboru vlož řádek '13;Pavel Vomáčka'" nebo "INSERT INTO zakaznik...".
- Programátorská dokumentace
  - Poskytuje pohled na strukturu programu – všechny procedury, třídy (metody i atributy), proměnné, ... (ne algoritmy)
    - U všech procedur, funkcí a metod stručně uveden význam všech parametrů a návratových hodnot, stejně jako popis funkce dané entity.
  - Součástí musí být diagram tříd, který popisuje kompletní objektovou strukturu programu – všechny vytvořené a použité třídy.
- Všechny identifikátory volte čitelně – tj. ne Metoda1 nebo Form8.
- Všechny bloky programu (procedury, funkce, metody) dělejte snesitelně dlouhé – maximálně 30 řádků, raději méně.