

Lecture 14

- Singleton pattern
- Fiaspel class diagram, sample code
- Course analysis

Singleton Pattern

- The Singleton pattern ensures that a class has only one instance and provides a global point of access to it.
- Examples:
 - There can be many printers in a system but there should only be one printer spooler.
 - There should be only one instance of a WindowManager (GrainWindowingSystem).
 - There should be only one instance of a filesystem.

Singleton Pattern

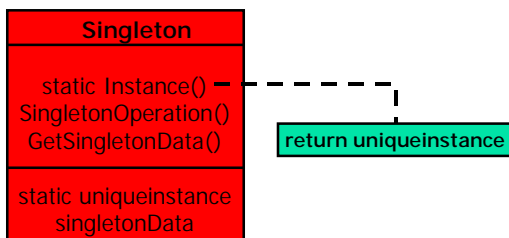
- How do we ensure that a class has only one instance and that the instance is easily accessible?
- A global variable makes an object accessible, but does not keep you from instantiating multiple objects.
- A better solution is to make the class itself responsible for keeping track of its sole instance. The class ensures that no other instance can be created (by intercepting requests to create new objects) and it provides a way to access the instance.

Singleton Pattern

Use the Singleton pattern when

- There must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
- When the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code.

Singleton Structure



Singleton Participants

- Singleton
 - Defines an Instance operation that lets clients access its unique instance. Instance is a class operation (static member function in C++)
 - May be responsible for creating its own unique instance
- Client
 - Accesses a Singleton instance solely through Singleton's Instance operation.

Singleton Consequences

- Controlled access to sole instance
Because the Singleton class encapsulates its sole instance, it can have strict control over how and when clients access it.
- Reduced name space
The Singleton pattern is an improvement over global variables. It avoids polluting the name space with global variables that store sole instances.

Singleton Consequences

- Permits refinement of operations and representations
The Singleton class may be subclassed and it is easy to configure an application with an instance of this extended class at run-time.
- More flexible than class operations
An alternative is to use static member functions in C++. However it is difficult to change the design to allow more than one instance of a class and static member functions in C++ are never virtual, so subclasses can not override polymorphically.

Singleton Implementation

- Ensuring a unique instance
The Singleton pattern makes the sole instance a normal instance of a class, but that class is written so that only one instance can ever be created. A common way to do this is to hide the operation that creates the instance behind a static class operation that guarantees that only one instance is created.

Singleton Sample Code

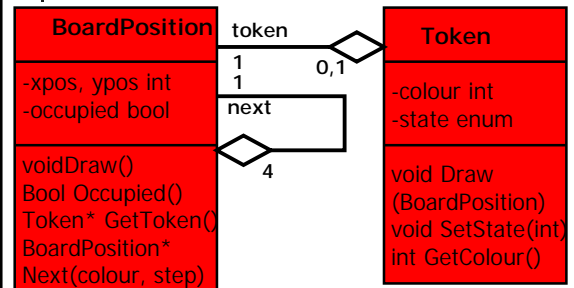
```
class Singleton {
public:
    static Singleton* Instance();
    // clients access the Singleton exclusively through
    // the Instance() member function
protected:
    Singleton();
    // the constructor is protected, such that a client
    // which tries to instantiate a Singleton object gets
    // a compiler error
private:
    static Singleton* instance_;
};
```

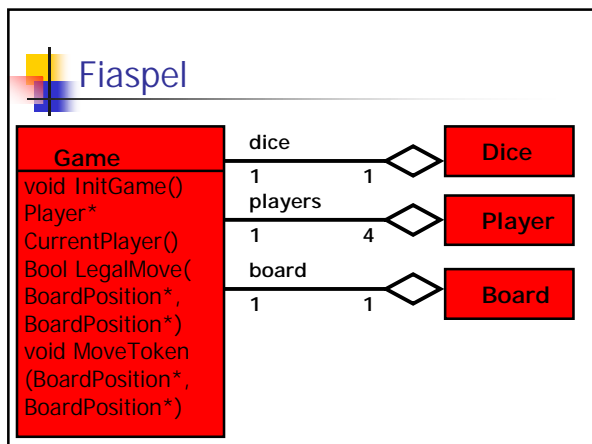
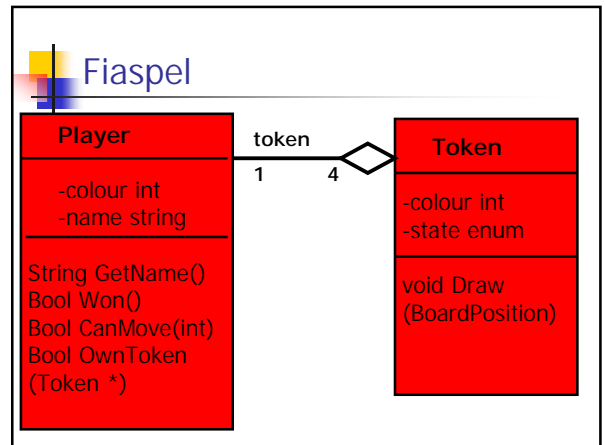
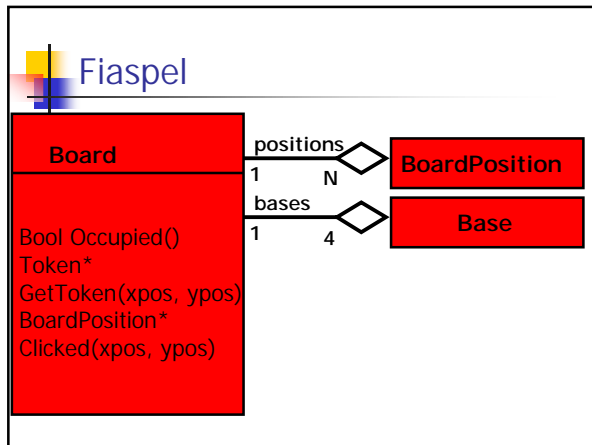
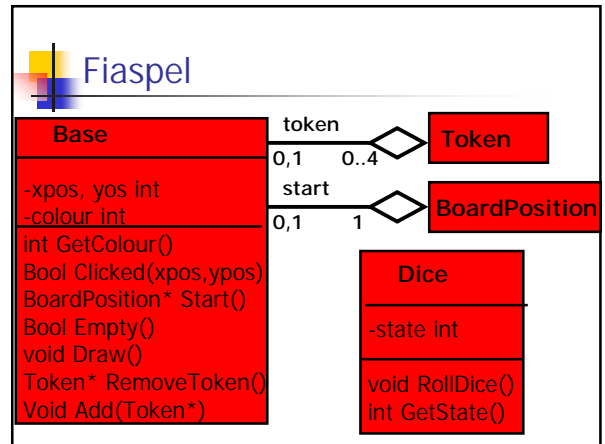
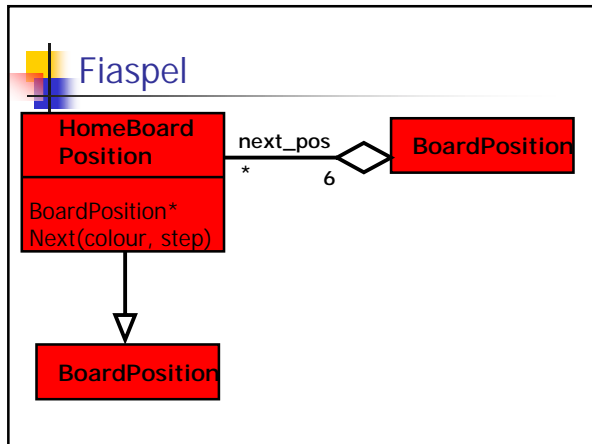
Singleton Sample Code

```
Singleton* Singleton::instance_ = 0;
// initialize static member data of class Singleton

Singleton* Singleton::Instance()
{
    if (instance_ == 0) // if not created yet
        instance_ = new Singleton; // create once
    return instance_;
}
```

Fiaspel





Sample Code

```

class Game
{
private:
    Board* board;
    vector<Player*> players;
    Dice* dice;
    Player* current_player;
public:
    bool LegalMove(Base* from, BoardPosition* to);
    bool LegalMove(BoardPosition* from, BoardPosition* to);
    Move(Base* from, BoardPosition* to);
    Move(BoardPosition* from, BoardPosition* to);
};
  
```

Sample Code

```
class BoardPosition
{
private:
    Token* token;
    vector<BoardPosition*> next;
    bool occupied;
public:
    bool Occupied() { return occupied;};
    void RemoveToken() { occupied=false; token=0;};
    Token* GetToken(){ return token;};
    AddToken(Token *new_token) {occupied=true; token=new_token;};
    virtual BoardPosition* Next(int colour, int steps) ;
    virtual BoardPosition* Next(int colour);
};
```

Sample Code

```
class HomeBoardPosition : public BoardPosition
{
private:
    vector<BoardPosition*> next_pos; // holds next board
    // positions for every possible value of the dice
public:
    virtual BoardPosition* Next(int colour, int steps);
    virtual BoardPosition* Next(int colour);
};
```

Sample Code

```
BoardPosition* BoardPosition::Next(int colour)
{
    return tmp->next[colour];
};

BoardPosition* BoardPosition::Next(int colour, int steps)
{
    BoardPosition* tmp=this;
    for (int i=0; i<steps; i++)
        tmp=tmp->Next(colour);
    return tmp;
};
```

Sample Code

```
BoardPosition* HomeBoardPosition::Next(int colour, int
steps)
{
    return next[steps-1];
};
BoardPosition* HomeBoardPosition::Next(int colour)
{
    return next[0];
};
```

Sample Code

```
Bool Game::LegalMove(BoardPosition* from, BoardPosition* to)
{
    if (! from->Occupied())
        return false;
    if (! from->GetToken()->GetColour()==CurrentPlayer()->GetColour())
        return false;
    if (to->GetToken()->GetColour()==CurrentPlayer()->GetColour())
        return false;
    if (from->Next(CurrentPlayer()->GetColour(),dice->GetState()) == to)
        return true;
    else
        return false;
}
```

Sample Code

```
Bool Game::LegalMove(Base* from, BoardPosition* to)
{
    if (from->Empty())
        return false;
    if (! from->GetColour()==CurrentPlayer()->GetColour())
        return false;
    if (to->GetToken()->GetColour()==CurrentPlayer()->GetColour())
        return false;
    if (from->Start()== to)
        return true;
    else
        return false;
}
```

Sample Code

```
enum token_states {inbase, onboard, home};

void Game::Move(BoardPosition* from, BoardPosition* to)
{
    Token *tmp=from->GetToken();
    from->RemoveToken();

    if (to->Occupied())
    {
        Token* kicked=to->GetToken();
        board->GetBase(kicked->GetColour())->Add(kicked);
        kicked->SetState(inbase);
    }
    to->AddToken(tmp);
}
```

Course Analysis

- Fill out the questionnaire on the course webpage
- Use the comment boxes for suggestions, complaints, negative and positive aspects of the course.

Course Analysis

- Do you think the course was
 - Easy
 - Medium
 - Difficult
- How was the relation in the lecture between object oriented programming and design and C++ language specific parts.
 - Too much C++
 - Right balance
 - Too much object oriented design

Course Analysis

- Do you think the course was interesting and useful for you?
 - Yes
 - Partially useful
 - No
- Do you think your previous knowledge (e.g. programming experience in C) was sufficient for this course?
 - Yes
 - Somewhat
 - No

Course Analysis

- What do you think about the course literature?
 - Stroustrup : The C++ Programming Language
 - Lippman : C++ Primer
 - Lafore : Object Oriented Prog. In C++
 - Gamma : Design Patterns

Course Analysis

- What do you think about the additional course material (e.g. slides, copies from books, quick references, tutorials, software tools)?
 - Useful
 - Partially useful
 - Useless or incomplete

Course Analysis

- What do you think about the lectures?
 - Pedagogics
 - Good
 - Acceptable
 - Unacceptable
 - Presentation, slides, questions
 - Good
 - Acceptable
 - Unacceptable

Course Analysis

- What do you think of the exercise hours
 - Contents
 - Useful
 - Partially useful
 - Not useful at all
 - Pedagogics , assistent
 - Competent
 - Partially Competent
 - Incompetent

Course Analysis

- What do you think about the lab hours
- Help by assistents
 - Good
 - Acceptable
 - Unacceptable
- Availability, number of hours, waiting time
 - Good
 - Acceptable
 - Unacceptable

Course Analysis

- Did you feel that you got enough and competent help in general, help beside the labs, pointers to reading, hints, tips
 - Good
 - Acceptable
 - Unacceptable

Course Analysis

- What do you think about the lab assignments
- Difficulty
 - Easy
 - Suitable
 - Difficult
- Programming tasks
 - Interesting
 - Partially interesting
 - Uninteresting

Course Analysis

- Which lab assignment did you like best?
 - Lab 2 menu system
 - Lab 3 drawing program
 - Lab 4 Fia game
- Which lab assignment did you like the least?
 - Lab 2 menu system
 - Lab 3 drawing program
 - Lab 4 Fia game

Course Analysis

- How much time did you spend on the labs in total?
 - Less than 60 hours
 - 60-120 hours
 - More than 120 hours
- What do you think about the examinations?
 - Fair
 - Mostly fair
 - Unfair

Course Analysis

- What is the percentage of your study time this semester that you spend on this course?
 - Less than 25%
 - 25-50%
 - More than 50%
- Do you think that 4 points for the course are
 - not enough
 - enough
 - too many

Course Analysis

- How would rate the lectures?
 - Very good
 - Good
 - Acceptable
 - Barely acceptable
 - Bad

Course Analysis

- How would rate the course in general?
 - Very good
 - Good
 - Acceptable
 - Barely acceptable
 - Bad

Course Analysis

- Was it a problem for you that the lectures were taught in English?
 - Yes
 - Somewhat of a problem
 - No

Course Analysis

- What did you like best about the course?
- What did you like least about the course?