INF011 – Padrões de Projeto

02 – *Creational Patterns*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Creational Patterns

- Abstraem o processo de instanciação de um ou vários objetos
- Tornam o sistema independente de como os objetos são criados, compostos e representados
- Um *class creational pattern* usa herança para variar a classe que é instanciada
- Um *object creational pattern* delega o processo de instanciação para outro objeto
- Se tornam importantes à medida em que o sistema passa a depender mais de composição e menos de herança

Creational Patterns

- Com a composição comportamentos *hard-coded* dão lugar a um conjunto de comportamentos fundamentais que podem ser compostos em comportamentos mais complexos
- Pontos comuns aos *creational patterns*:
  - Encapsulam conhecimento sobre qual classe concreta o sistema usa
  - Ocultam como instâncias dessas classes são criadas e integradas
- Tudo o que o sistema conhece sobre os objetos são as interfaces por eles implementadas, definidas através de classes abstratas

Creational Patterns

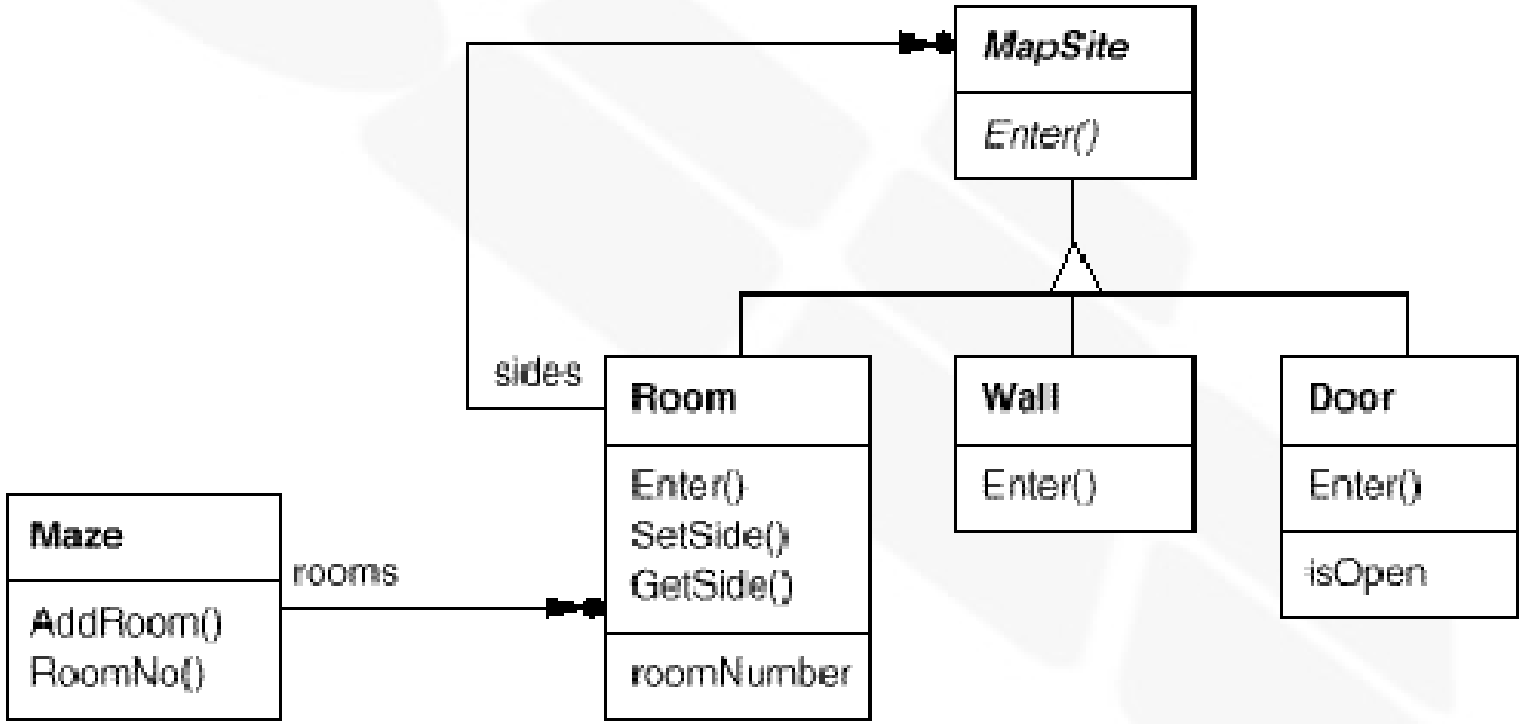
- Trazem flexibilidade sobre o que é criado, por quem, como e quando
- Permitem configurar o sistema para o uso de objetos "produto" com estrutura e funcionalidades variadas
- A configuração pode ser estática ou dinâmica
- *Creational patterns* alternativos: *Prototype* ou *Abstract Factory*
- *Creational patterns* complementares: *Builder* pode usar um dos outros *patterns* para implementar quais componentes são criados. *Prototype* pode usar um *Singleton* na sua implementação

Creational Patterns

- Os cinco *creational patterns* serão estudados com base em um exemplo comum a todos: um jogo de labirinto
- O foco é em como os labirintos são criados
- Define-se um labirinto como uma coleção de salas
- Uma sala conhece seus quatro componentes formadores: outra sala, uma parede ou uma parede com porta para outra sala

Creational Patterns

- Classes utilizadas em todos os exemplos:



Creational Patterns

- Classes utilizadas em todos os exemplos:

```
class Room : public MapSite {
public:
    Room(int roomNo);

    MapSite* GetSide(Direction) const;
    void SetSide(Direction, MapSite*);

    virtual void Enter();

private:
    MapSite* _sides[4];
    int _roomNumber;
};
```

Creational Patterns

- Classes utilizadas em todos os exemplos:

```
class Wall : public MapSite {
public:
    Wall();

    virtual void Enter();
};

class Door : public MapSite {
public:
    Door(Room* = 0, Room* = 0);

    virtual void Enter();
    Room* OtherSideFrom(Room*);

private:
    Room* _room1;
    Room* _room2;
    bool _isOpen;
};
```

Creational Patterns

- Classes utilizadas em todos os exemplos:

```
class Maze {
public:
    Maze();

    void AddRoom(Room*);
    Room* RoomNo(int) const;
private:
    // ...
};
```

Creational Patterns

- Classes utilizadas em todos os exemplos:

```
Maze* MazeGame::CreateMaze () {
    Maze* aMaze = new Maze;
    Room* r1 = new Room(1);
    Room* r2 = new Room(2);
    Door* theDoor = new Door(r1, r2);

    aMaze->AddRoom(r1);
    aMaze->AddRoom(r2);

    r1->SetSide(North, new Wall);
    r1->SetSide(East, theDoor);
    r1->SetSide(South, new Wall);
    r1->SetSide(West, new Wall);

    r2->SetSide(North, new Wall);
    r2->SetSide(East, new Wall);
    r2->SetSide(South, new Wall);
    r2->SetSide(West, theDoor);

    return aMaze;
}
```

Creational Patterns

- O código-fonte anterior é inflexível
- Ele define, de forma *hard-coded*, o *layout* do labirinto
- Para mudar o *layout* é necessário mudar ou sobrepor este método → pouco reuso e falhas em potencial
- Os *creational patterns* tornam este código mais flexível, não necessariamente menor
- Torna fácil mudar as classes que definem os componentes do labirinto
- Como mudar *CreateMaze* para suportar *DoorNeedingSpell* e *EnchantedRoom*, mantendo o mesmo *layout* ?

Creational Patterns

- O problema é a dependência explícita dos tipos concretos:
  - *CreateMaze* pode chamar métodos abstratos ao invés de construtores. Sub-classes de *MazeGame* implementariam estes métodos abstratos *(Factory Method)*
  - *CreateMaze* pode receber um objeto como parâmetro e o utilizar para criar salas, paredes e portas. Diferentes objetos criariam diferentes elementos *(Abstract Factory)*
  - *CreateMaze* pode receber um objeto como parâmetro que cria o labirinto completo através de métodos para adicionar salas, paredes e portas ao labirinto. Herança pode ser utilizada para mudar as partes ou o *layout (Builder)*

Creational Patterns

- O problema é a dependência explícita dos tipos concretos:
  - *CreateMaze* pode ter protótipos de salas, paredes e portas como parâmetros. Este protótipos são copiados para compor o labirinto. Diferentes labirintos podem ser obtidos com a utilização de protótipos diferentes (*Prototype*)
- O padrão *Singleton* pode:
  - Garantir a existência de somente um labirinto por jogo e que todos os objetos do jogo tenham pronto acesso a ele, sem utilizar variáveis ou funções globais
  - Possibilitar a extensão ou troca do labirinto sem alterações no código existente

INF011 – Padrões de Projeto

02 – *Creational Patterns*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas