Creational design patterns abstract the instantiation process. They help make a system independent of how its objects are created, composed, and represented.

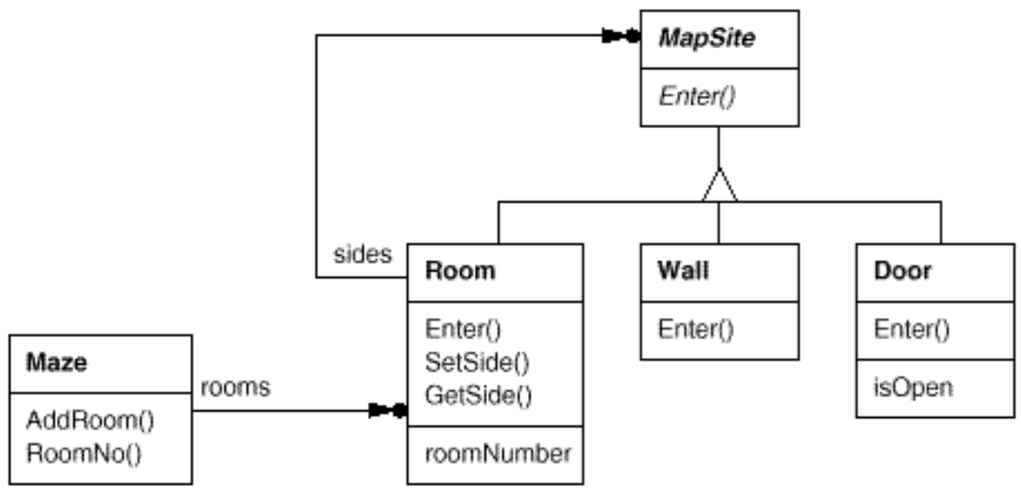
- A class creational pattern uses inheritance to vary the class that's instantiated. (static, specified at compile-time)
- An object creational pattern will delegate instantiation to another object. (dynamic, specified at run-time)

Two recurring themes:

- the all encapsulate knowledge about which concrete classes the system uses.
- the hide how instances of these classes are created and put together.

Example: A Maze game

we'll just focus on how mazes get created.




```
enum Direction { North, South, East, West };

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;
}
```

To make it simpler?

the Room constructor could initialize the sides with walls ahead of time. ???

- That just moves the code somewhere else.
- The real problem with this member function isn't its size but its inflexibility.
- It hard-codes the maze layout.

How?

- Factory Method
- Abstract Factory
- Builder
- Prototype