Sample Code

The function `CreateMaze` (page 84) builds and returns a maze. One problem with this function is that it hard-codes the classes of maze, rooms, doors, and walls. We'll introduce factory methods to let subclasses choose these components.

First we'll define factory methods in `MazeGame` for creating the maze, room, wall, and door objects:

```
class MazeGame {
public:
    Maze* CreateMaze();

// factory methods:

    virtual Maze* MakeMaze() const
        { return new Maze; }
    virtual Room* MakeRoom(int n) const
        { return new Room(n); }
    virtual Wall* MakeWall() const
        { return new Wall; }
    virtual Door* MakeDoor(Room* r1, Room* r2) const
        { return new Door(r1, r2); }
};
```

Each factory method returns a maze component of a given type. `MazeGame` provides default implementations that return the simplest kinds of maze, rooms, walls, and doors.

Now we can rewrite `CreateMaze` to use these factory methods:

```
Maze* MazeGame::CreateMaze () {
    Maze* aMaze = MakeMaze();

    Room* r1 = MakeRoom(1);
    Room* r2 = MakeRoom(2);
    Door* theDoor = MakeDoor(r1, r2);

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

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

    r2->SetSide(North, MakeWall());
    r2->SetSide(East, MakeWall());
    r2->SetSide(South, MakeWall());
    r2->SetSide(West, theDoor);
```