

* **Creational** patterns:
  + Abstract the object creation process.
  + E.g., Singleton, Abstract Factory, …
* **Structural** patterns:
  + Composition of classes or objects
  + Decouple interface and implementation of classes
  + E.g., Adapter, Façade, Composite, …
* **Behavioral** patterns:
  + Deal with dynamic interactions among societies of objects
    - How they distribute responsibility and how they communicate
  + E.g., Iterator, Observer, Visitor, …
* Benefits:
* Separate the responsibility of object creation process
  + Isolate clients from implementation classes
* Makes changing product families easy
* Promotes the consistency between products

class RegularMazeFactory **extends MazeFactory** {

public Room makeRoom () {

return new Room();

}

public Door makeDoor (Room r1, Room r2) {

return new Door(r1, r2);

}

public Wall makeWall () {

return new Wall();

}

}

abstract class MazeFactory {

public abstract Room **makeRoom** ();

public abstract Door **makeDoor** (Room r1, Room r2);

public abstract Wall **makeWall** ();

}

public class MazeGame {

**MazeFactory fac**=null;

public MazeGame(MazeFactory fac) {

this.fac=fac;}

public Maze createMaze() {

Maze aMaze= new Maze();

Room r1 = **fac.makeRoom**();

Room r2 = **fac.makeRoom**();

Door theDoor = **fac.makeDoor**(r1,r2);

…

}

}

* GoF design patterns
  + Singleton, Adapter, Abstract Factory
* GRASP
  + Information Expert, Creator, Low Coupling, High Cohesion, Controller, Pure Fabrication, Polymorphism, Indirection

class EnchantedMazeFactory **extends MazeFactory** {

public Room makeRoom () {

return new EnchantedRoom();

}

public Door makeDoor (Room r1, Room r2) {

return DoorWithSpell(r1, r2);

}

public Wall makeWall () {

return new Wall();

}

}