13__20-08-2022

Saturday, 20 August 2022 8:08 PM

→ Abstract factory design pattern

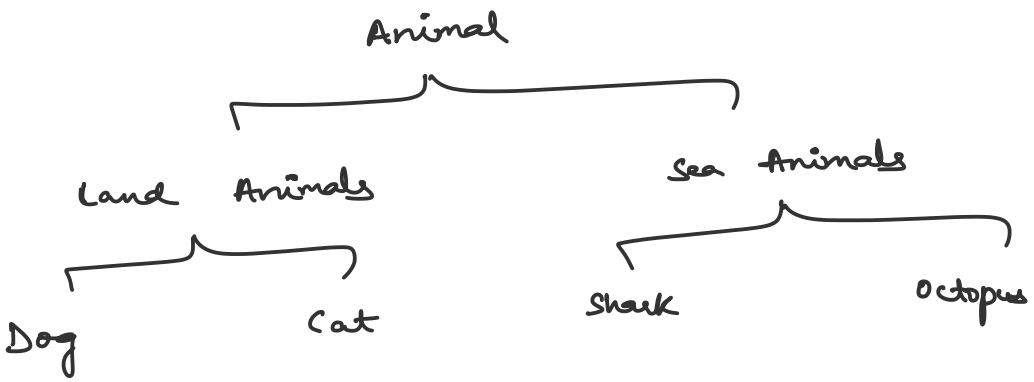
→ Advantages

→ Usages

→ Implementation Guidelines

→ Difference between factory method and abstract factory pattern.

Abstract factory



Abstract factory is also called as factories

of factories or kit.

According to Gang of four —

The Abstract factory pattern provides a way to encapsulate a group of individual factories that have a common theme without specifying the concrete classes.

Abstract factory pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes.

Animal factory → super factory

Land Animal Factory

Sea Animal Factory

Abstract factory works around a super-factory which creates other factories.

In Abstract factory each generated factory can give the objects as per the factory pattern.

Implementation guidelines

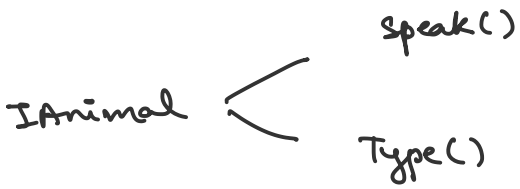
1. It should be used in the applications where we have need to create multiple families of objects or products.

2. we need to use only one of the subset of the families of objects at a given point of time.

3. we want to hide the implementations of the families of product by decoupling the implementation of each of these products.

Steps to create abstract factory

Step 1 - Create an interface.



concrete classes implementing

Step 2 - Create ... of

the same interface.

Step 3 - Create an abstract class to get factories.

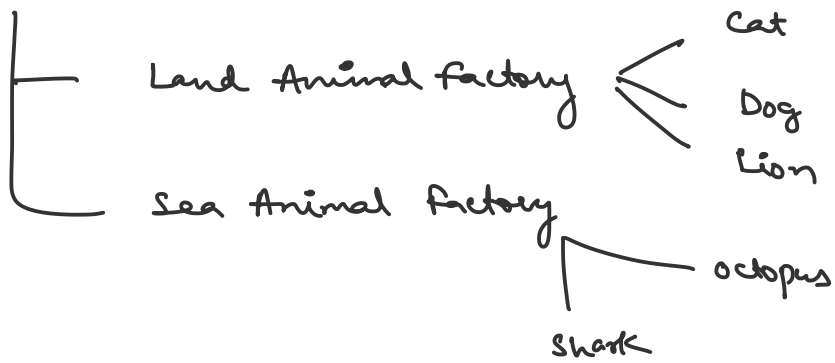
AnimalFactory

→ returning specific factories

- Sea Animal factory
- Land Animal factory

Step 4 - Create factory classes extending Abstract factory to generate objects of ... the information.

concrete class based on



Step 5 - Create a factory generator class to get factories by passing an information.

```
public interface IAnimal {
   public String speak();
   public String type();
}
```

```
public class Cat implements IAnimal {
   @Override
   public String speak() {
      return "Meow";
   }

   @Override
   public String type() {
      return "Cat";
   }
}
```

```
public class Dog implements IAnimal {
   @Override
   public String speak() {
      return "Bark";
   }

   @Override
   public String type() {
      return "Dog";
   }
}
```

```
public class Lion implements IAnimal {
   @Override
   public String speak() {
      return "Roar";
   }

   @Override
   public String type() {
      return "Lion";
   }
}
```

```
public class Shark implements IAnimal {
   @Override
   public String speak() {
      return "Cannot speak";
   }

   @Override
   public String type() {
      return "Shark";
   }
}
```

```
public class Octopus implements IAnimal {
   @Override
   public String speak() {
      return "Squawck";
   }

   @Override
   public String type() {
      return "Octopus";
   }
}
```

```
public abstract class AnimalFactory {
   public abstract IAnimal getAnimal(String animalType);

   public static AnimalFactory createAnimalFactory(String factoryType)
   {
      if(factoryType == "Sea")
      {
         return new SeaAnimalFactory();
      }
      else if(factoryType == "Land")
      {
         return new LandAnimalFactory();
      }
      else
      {
         System.out.println("Factory not supported!!");
         return null;
      }
   }
}
```

```
public class LandAnimalFactory extends AnimalFactory {
   @Override
   public IAnimal getAnimal(String animalType)
   {
```

```
        if(animalType == "Cat")
        {
            return new Cat();
        }
        else if(animalType == "Dog")
        {
            return new Dog();
        }
        else if(animalType == "Lion")
        {
            return new Lion();
        }
        else
        {
            System.out.println("Land Animal type not supported!!");
            return null;
        }
    }
}
```

```
public class SeaAnimalFactory extends AnimalFactory {
    @Override
    public IAnimal getAnimal(String animalType) {
        if(animalType == "Shark")
        {
            return new Shark();
        }
        else if(animalType == "Octopus")

        {
            return new Octopus();
        }
        else
        {
            System.out.println("Sea Animal type not supported!!");
            return null;
        }
    }
}
```

```
public class Program {
```

```
    public static void main(String[] args) {
        printDetails("abc", "xyz");
        printDetails("Land", "Lion");
        printDetails("Land", "Dog");
        printDetails("Land", "Cat");
        printDetails("Sea", "Octopus");
        printDetails("Sea", "Shark");
    }

    private static void printDetails(String factoryType, String animalType) {
        AnimalFactory animalFactory = AnimalFactory.createAnimalFactory(factoryType);

        IAnimal animal;
        if(animalFactory != null)
        {
            animal = animalFactory.getAnimal(animalType);

            {
                if(animal != null)
                {
                    System.out.println("Animal factory type : " + animal.type());
                    System.out.println("Animal speak : " + animal.speak());
                }
            }
        }
        System.out.println();
    }
}
```

Output:

```
Factory not supported!!

Animal factory type : Lion
Animal speak : Roar

Animal factory type : Dog
Animal speak : Bark

Animal factory type : Cat
Animal speak : Meow

Animal factory type : Octopus
Animal speak : Squawck

Animal factory type : Shark
```

Animal speak : Cannot speak