1. 单例模式：https://www.cnblogs.com/hupp/p/4487521.html（单例模式的五种写法）

保证一个类仅有一个实例，并提供一个访问它的全局访问点。

常用写法:

这是一种懒汉模式。

```
public class Singleton {
    /*
     * 持有私有静态实例，防止被引用，此处赋值为null，目的是实现延迟加载。
     */
     private static Singleton instance = null;

     /*私有构造方法，防止被实例化*/
     private Singleton() {}

     /*懒汉式，静态工程方法，创建实例*/
     public static Singleton getInstance(){
         if (instance == null) {
            instance = new Singleton();
        }
         return instance;
     }
}

/*
 * 优点：延迟加载（只有需要时才去加载），适合单线程操作。
 * 缺点：线程不安全，在多线程中容易出现不同步的情况，如在数据库对象进行频繁读写操作时候。
 */
```

为了解决线程不安全的情况，提出了其他方法。

1. 内部类的方式

```
public class Singleton {

    /*
     * 内部类实现单例模式
     * 延迟加载，减少内存消耗
     */
    private Singleton(){}

    private static class SingletonInner{

         private static Singleton instance = new Singleton();
    }

    public static Singleton getInstance(){

        return SingletonInner.instance;
    }
}

/*延迟加载，线程安全（java中class加载时互斥），也减少了内存消耗，推荐使用内部类方式*/
```

2. 双检查锁的方式

```java
public class SingleModuleDoubleLock {

    private static SingleModuleDoubleLock singleton;

    private SingleModuleDoubleLock() {

    }

    public static SingleModuleDoubleLock getSingle() {
        if(singleton == null) {
            synchronized (SingleModuleDoubleLock.class) {
                if(singleton == null)
                    singleton = new SingleModuleDoubleLock();
            }
        }
        return singleton;
    }

}
```

建立工厂模式的原因是让用户的代码和特定类的子类代码解耦，工厂方法使用户不必知道它所使用的对象是怎样被创建的，只需要知道该对象有哪些方法即可。对于工厂模式需要有两个概念，一是产品、二是工厂。

2.简单工厂模式：

bingfa.java AThread.java test.java *Singleton.java

```java
package SimpleFactory;

public class SimpleFactory {
   public static void  main(String [] args) {
    Factory factory = new Factory();
    factory.produce("PRO5").run();
    factory.produce("PRO6").run();
 }
}

//这是一个抽象产品
interface MeizuPhone{
    void run();
}

/*
 * 下面两个为具体产品
 */
class PRO5 implements MeizuPhone{

    public void run() {
      System.out.println("我是一台PRO5");
    }

}

class PRO6 implements MeizuPhone{

    public void run() {
        System.out.println("我是一台PRO6");
    }

}

/*一个简单工厂*/
class Factory{
    MeizuPhone produce(String product) {
        if("PRO5".equals(product)){
            return new PRO5();
        }else if ("PRO6".equals(product)) {
            return new PRO6();
        }
        return null;
    }
}
```

简单工厂是不易维护的，如果需要添加新的产品，则整个系统都需要修改，如果我们需要添加PRO7、PRO8等产品，直接在工程类中添加即可。

```java
package FactoryMethod;

public class FactoryMethod {
   public static void main(String [] args) {
       IFactory bigFactory;
       bigFactory = new SmallFactory();
       bigFactory.produce().run();
       bigFactory = new BigFactory();
       bigFactory.produce().run();
   }
}

interface MeizuPhone{
    void run();
}

class PRO5 implements MeizuPhone{

    public void run() {
        System.out.println("我是一台PRO5");
    }

}

class MX5 implements MeizuPhone{

    public void run() {
        System.out.println("我是一台MX5");
    }

}

interface IFactory{
    MeizuPhone produce();
}

class BigFactory implements IFactory{

    public MeizuPhone produce() {
        return new PRO5();
    }
}

class SmallFactory implements IFactory{

    public MeizuPhone produce() {
      return new MX5();
    }

}
```

与简单工厂间的取舍：工厂方法模式和简单工厂模式在定义上的不同是非常明显的。工厂方法模式的核心是一个抽象工厂类，而不像简单工厂模式，把核心放在一个实体类上。工厂方法模式可以允许很多实的工厂类从抽象工厂类继承下来，从而可以在实际上成为多个简单工厂模式的综合，从而推广了简单工厂模式。