深圳大学
Shenzhen University

第13-1讲

享元模式

软件体系结构与设计模式

Software Architecture & Design Pattern

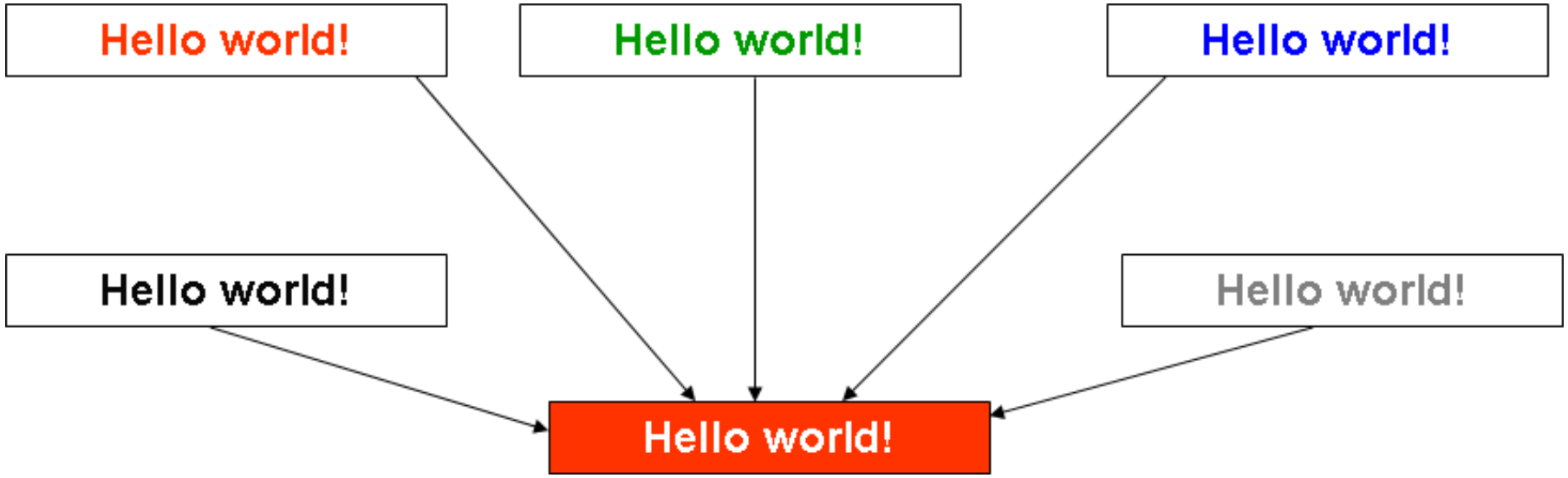
深圳大学计算机与软件学院

主要内容

- **1享元模式**
  - 享元模式动机与定义
  - 享元模式结构与分析
  - 享元模式实例与解析
  - 享元模式效果与应用

1 享元模式

- **享元模式动机**
  - 字符串享元对象示意图

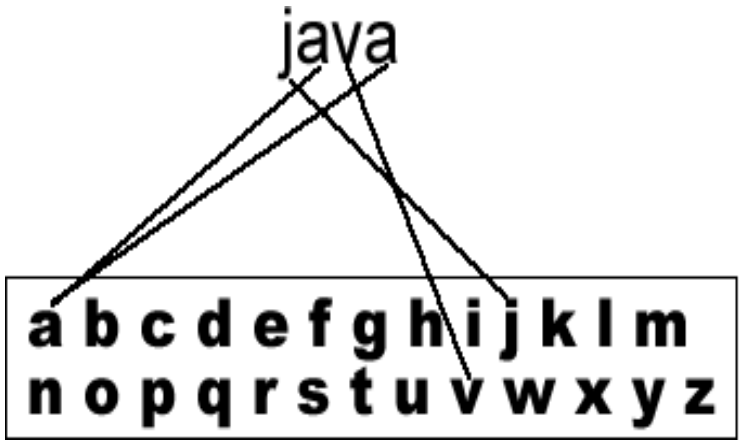


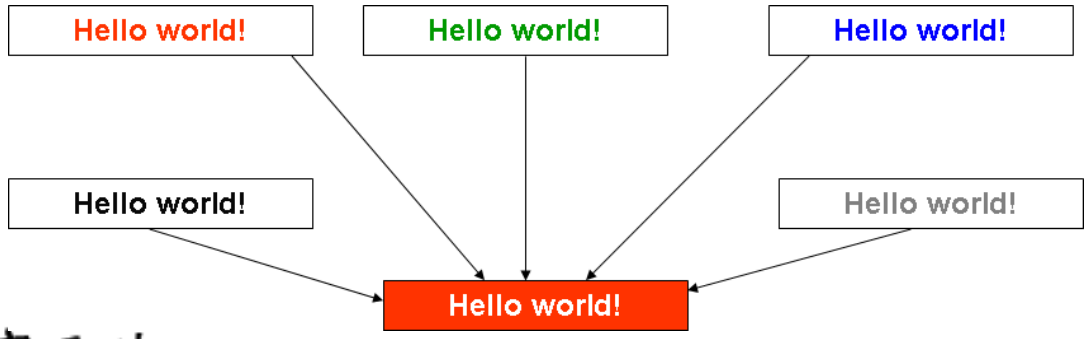
享元模式动机

- 如果一个软件系统在运行时所创建的相同或相似对象数量太多，将导致运行代价过高，带来系统资源浪费、性能下降等问题
- 如何避免系统中出现大量相同或相似的对象，同时又不影响客户端程序通过面向对象的方式对这些对象进行操作呢？→ **享元模式**

享元模式动机

- **享元模式**：通过共享技术实现相同或相似对象的重用
- **享元池(Flyweight Pool)**：存储共享实例对象的地方





享元池

享元模式定义

- 对象结构型模式

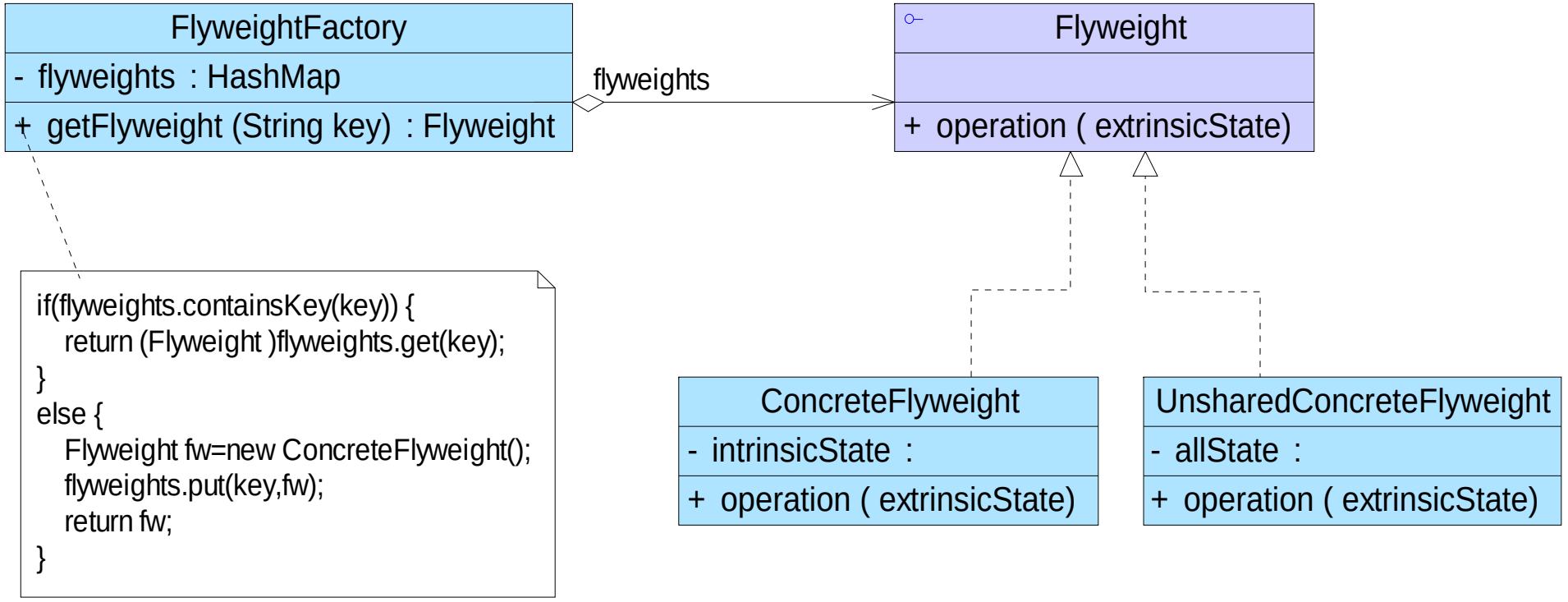
享元模式：运用共享技术有效地支持大量细粒度对象的复用。

Flyweight Pattern: Use sharing to support large numbers of fine-grained objects efficiently.

- 又称为轻量级模式
- 要求能够被共享的对象必须是细粒度对象

享元模式结构

- **享元模式包含如下角色：**
  - Flyweight: 抽象享元类
  - ConcreteFlyweight: 具体享元类
  - UnsharedConcreteFlyweight: 非共享具体享元类
  - FlyweightFactory: 享元工厂类



享元模式结构与分析

- **内部状态(Intrinsic State)**：存储在享元对象内部并且不会随环境改变而改变的状态，内部状态可以共享（例如：字符的内容）
- **外部状态(Extrinsic State)**：随环境改变而改变的、不可以共享的状态。享元对象的外部状态通常由客户端保存，并在享元对象被创建之后，需要使用的时候再传入到享元对象内部。一个外部状态与另一个外部状态之间是相互独立的（例如：字符的颜色和大小）

享元模式结构与分析

- **实现过程**
  - (1) 将具有相同内部状态的对象存储在享元池中，享元池中的对象是可以实现共享的
  - (2) 需要的时候将对象从享元池中取出，即可实现对象的复用
  - (3) 通过向取出的对象注入不同的外部状态，可以得到一系列相似的对象，而这些对象在内存中实际上只存储一份

享元模式分析

- **具体享元类示例代码：**

```
public class ConcreteFlyweight extends Flyweight {
    //内部状态intrinsicState作为成员变量，同一个享元对象其内部状态是一致的
    private String intrinsicState;
    public ConcreteFlyweight(String intrinsicState) {
        this.intrinsicState = intrinsicState;
    }

    //外部状态extrinsicState在使用时由外部设置，不保存在享元对象中，即使是
//同一个对象，在每一次调用时可以传入不同的外部状态
    public void operation(String extrinsicState) {
        //实现业务方法
    }
}
```

享元模式分析

- **享元工厂类示例代码：**

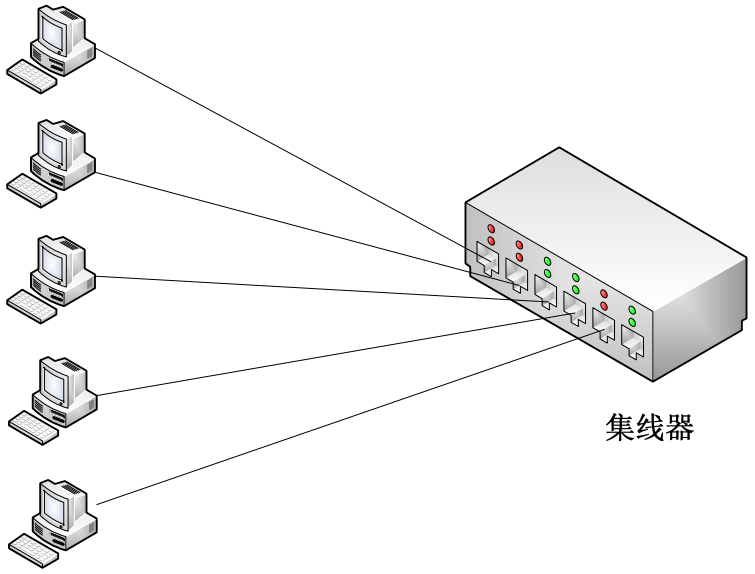
```
public class FlyweightFactory {
    //定义一个HashMap用于存储享元对象，实现享元池
    private HashMap flyweights = new HashMap();

    public Flyweight getFlyweight(String key) {
      //如果对象存在，则直接从享元池获取
      if (flyweights.containsKey(key)) {
          return (Flyweight)flyweights.get(key);
       }
       //如果对象不存在，先创建一个新的对象添加到享元池中，然后返回
       else {
          Flyweight fw = new ConcreteFlyweight();
          flyweights.put(key,fw);
          return fw;
       }
    }
}
```

享元模式实例

共享网络设备（无外部状态）：实例说明

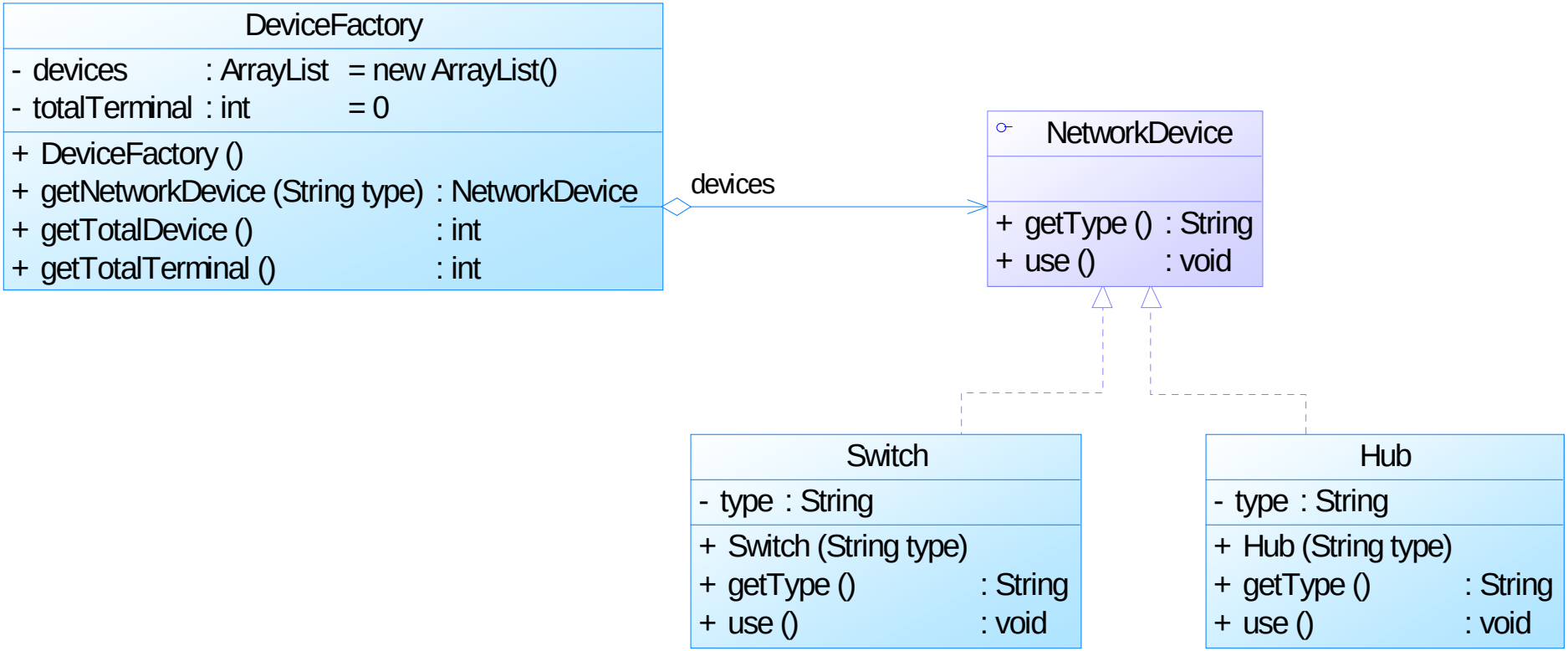
- 很多网络设备都是支持共享的，如交换机、集线器等，多台终端计算机可以连接同一台网络设备，并通过该网络设备进行数据转发，如图所示，现用享元模式模拟共享网络设备的设计原理。



享元模式实例与解析

享元模式实例

共享网络设备（无外部状态）：参考类图



享元模式实例与解析

- **享元模式实例**
  - **共享网络设备（无外部状态）：参考代码**
- DesignPatterns之flyweight.nonextrinsic包

NetworkDevice.java

```
1  package flyweight.nonextrinsic;
2
3  public interface NetworkDevice
4  {
5      public String getType();
6      public void use();
7  }
```

DeviceFactory.java

```java
import java.util.*;
public class DeviceFactory
{
    private ArrayList devices = new ArrayList();
    private int totalTerminal=0;

    public DeviceFactory()
    {
        NetworkDevice nd1=new Switch("Cisco-WS-C2950-24");
        devices.add(nd1);
        NetworkDevice nd2=new Hub("TP-LINK-HF8M");
        devices.add(nd2);
    }

    public NetworkDevice getNetworkDevice(String type)
    {
        if(type.equalsIgnoreCase("cisco"))
        {
            totalTerminal++;
            return (NetworkDevice)devices.get(0);
        }
        else if(type.equalsIgnoreCase("tp"))
        {
            totalTerminal++;
            return (NetworkDevice)devices.get(1);
        }
```

DeviceFactory.java

```java
        else
        {
            return null;
        }
    }

    public int getTotalDevice()
    {
        return devices.size();
    }

    public int getTotalTerminal()
    {
        return totalTerminal;
    }
}
```

Hub.java

```
3  public class Hub implements NetworkDevice
4  {
5      private String type;
6
7      public Hub(String type)
8      {
9          this.type=type;
10     }
11
12     public String getType()
13     {
14         return this.type;
15     }
16
17     public void use()
18     {
19         System.out.println("Linked by Hub, type is " + this.type);
20     }
21 }
```

Switch.java

```
3  public class Switch implements NetworkDevice
4  {
5      private String type;
6
7      public Switch(String type)
8      {
9          this.type=type;
10     }
11
12     public String getType()
13     {
14         return this.type;
15     }
16
17     public void use()
18     {
19         System.out.println("Linked by switch, type is " + this.type);
20     }
21 }
```

Client.java

```
2
3  public class Client
4  {
5      public static void main(String args[])
6      {
7          NetworkDevice nd1,nd2,nd3,nd4,nd5;
8          DeviceFactory df=new DeviceFactory();
9
10         nd1=df.getNetworkDevice("cisco");
11         nd1.use();
12
13         nd2=df.getNetworkDevice("cisco");
14         nd2.use();
15
16         nd3=df.getNetworkDevice("cisco");
17         nd3.use();
18
19         nd4=df.getNetworkDevice("tp");
20         nd4.use();
21
22         nd5=df.getNetworkDevice("tp");
23         nd5.use();
24
25         System.out.println("Total Device:" + df.getTotalDevice());
26         System.out.println("Total Terminal:" + df.getTotalTerminal());
27     }
28 }
```

享元模式实例

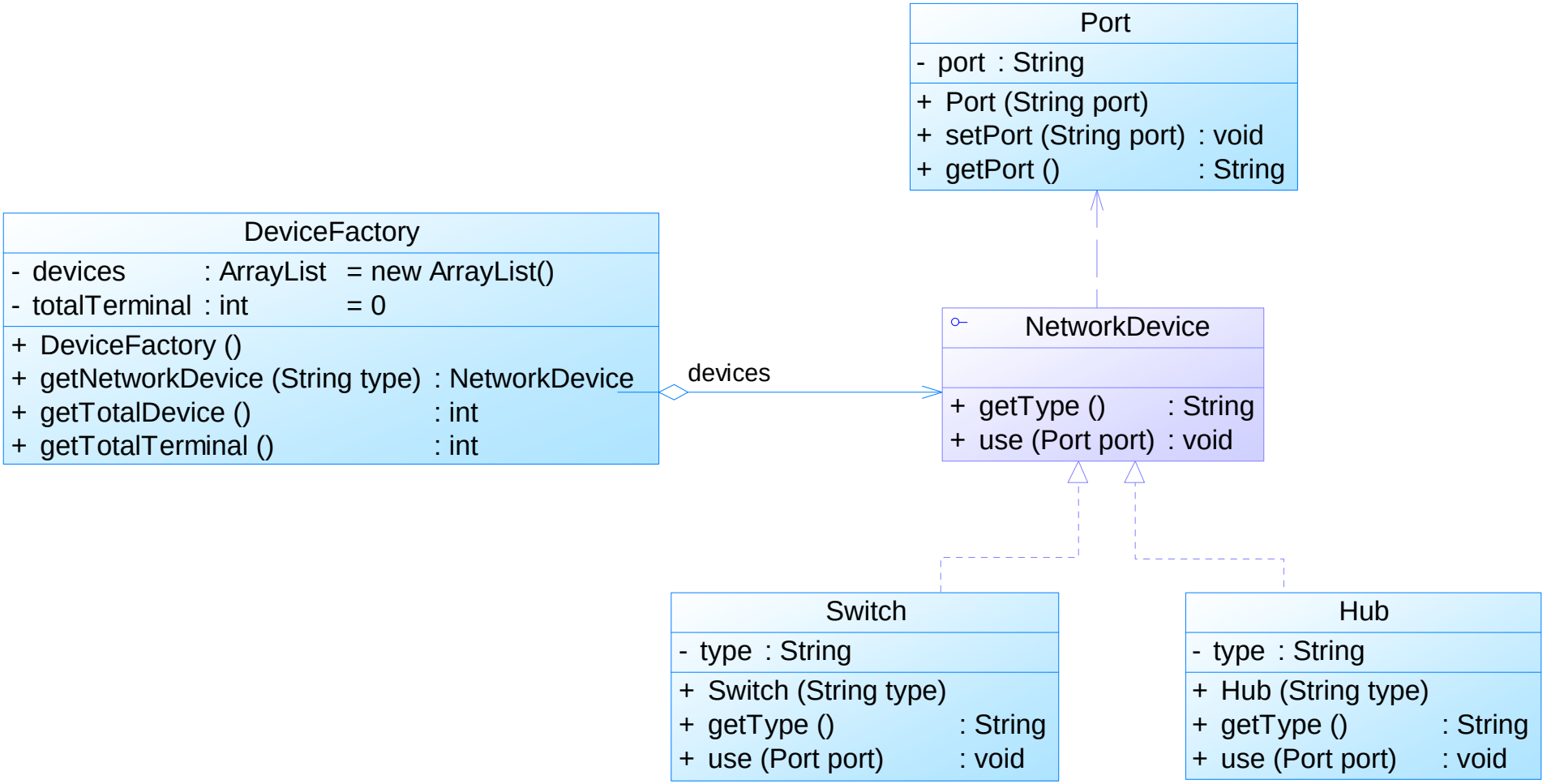
共享网络设备（有外部状态）：实例说明

- 虽然网络设备可以共享，但是分配给每一个终端计算机的端口(Port)是不同的，因此多台计算机虽然可以共享同一个网络设备，但必须使用不同的端口。我们可以将端口从网络设备中抽取出来作为外部状态，需要时再进行设置。

享元模式实例与解析

享元模式实例

共享网络设备（有外部状态）：参考类图



享元模式实例与解析

- **享元模式实例**
  - **共享网络设备（有外部状态）：参考代码**
- DesignPatterns之flyweight.extrinsic包

NetworkDevice.java

```
1  package flyweight.extrinsic;
2
3  public interface NetworkDevice
4  {
5      public String getType();
6      public void use(Port port);
7  }
```

DeviceFactory.java

```java
import java.util.*;
public class DeviceFactory
{
    private ArrayList devices = new ArrayList();
    private int totalTerminal=0;

    public DeviceFactory()
    {
        NetworkDevice nd1=new Switch("Cisco-WS-C2950-24");
        devices.add(nd1);
        NetworkDevice nd2=new Hub("TP-LINK-HF8M");
        devices.add(nd2);
    }

    public NetworkDevice getNetworkDevice(String type)
    {
        if(type.equalsIgnoreCase("cisco"))
        {
            totalTerminal++;
            return (NetworkDevice)devices.get(0);
        }
        else if(type.equalsIgnoreCase("tp"))
        {
            totalTerminal++;
            return (NetworkDevice)devices.get(1);
        }
```

DeviceFactory.java

```java
        else
        {
            return null;
        }
    }

    public int getTotalDevice()
    {
        return devices.size();
    }

    public int getTotalTerminal()
    {
        return totalTerminal;
    }
}
```

Port.java

```java
package flyweight.extrinsic;

public class Port
{
    private String port;

    public Port(String port)
    {
        this.port=port;
    }

    public void setPort(String port)
    {
        this.port=port;
    }

    public String getPort()
    {
        return this.port;
    }
}
```

Switch.java

```
1  package flyweight.extrinsic;
2
3  public class Switch implements NetworkDevice
4  {
5      private String type;
6
7      public Switch(String type)
8      {
9          this.type=type;
10     }
11
12     public String getType()
13     {
14         return this.type;
15     }
16
17     public void use(Port port)
18     {
19         System.out.println("Linked by switch, type is " +
20     this.type + ", port is " + port.getPort());
21     }
22 }
```

Hub.java

```java
package flyweight.extrinsic;

public class Hub implements NetworkDevice
{
    private String type;

    public Hub(String type)
    {
        this.type=type;
    }

    public String getType()
    {
        return this.type;
    }

    public void use(Port port)
    {
        System.out.println("Linked by Hub, type is "
    + this.type + ", port is " + port.getPort());
    }
}
```

Client.java

```
2
3  public class Client
4  {
5      public static void main(String args[])
6      {
7          NetworkDevice nd1,nd2,nd3,nd4,nd5;
8          DeviceFactory df=new DeviceFactory();
9
10         nd1=df.getNetworkDevice("cisco");
11         nd1.use(new Port("1000"));
12
13         nd2=df.getNetworkDevice("cisco");
14         nd2.use(new Port("1001"));
15
16         nd3=df.getNetworkDevice("cisco");
17         nd3.use(new Port("1002"));
18
19         nd4=df.getNetworkDevice("tp");
20         nd4.use(new Port("1003"));
21
22         nd5=df.getNetworkDevice("tp");
23         nd5.use(new Port("1004"));
24
25         System.out.println("Total Device:" + df.getTotalDevice());
26         System.out.println("Total Terminal:" + df.getTotalTerminal());
27     }
28 }
```

享元模式效果与应用

- **享元模式优点：**
  - 可以减少内存中对象的数量，使得相同或者相似的对象在内存中只保存一份，从而可以节约系统资源，提高系统性能
  - 外部状态相对独立，而且不会影响其内部状态，从而使得享元对象可以在不同的环境中被共享

享元模式效果与应用

- **享元模式缺点：**
  - 使得系统变得复杂，需要分离出内部状态和外部状态，这使得程序的逻辑复杂化
  - 为了使对象可以共享，享元模式需要将享元对象的部分状态外部化，而读取外部状态将使得运行时间变长

享元模式效果与应用

- **在以下情况下可以使用享元模式：**
  - 一个系统有大量相同或者相似的对象，造成内存的大量耗费
  - 对象的大部分状态都可以外部化，可以将这些外部状态传入对象中
  - 在使用享元模式时需要维护一个存储享元对象的享元池，而这需要耗费一定的系统资源，因此，在需要多次重复使用享元对象时才值得使用享元模式

谢谢