1. 内部类

概念：将一个类放在另一个类定义的内部

从外部类的非静态方法之外的任意位置创建某个内部类的对象，必须具体地指明这个对象的类型：OuterClassName.InnerClassName。

非静态内部类可以访问外部类的所有成员方法和成员属性，这是因为外部类的对象创建一个内部类对象时，此内部类会自动获取一个指向外部类的引用。（这里要注意对象生命周期，垃圾回收机制）

```
public class OuterClass {
    private static String name = "sfas";
    class InnerClass{
        public void test(){
            System.out.println(name);
        }
    }
    public static void main(String[] args) {
        InnerClass in = new OuterClass().new InnerClass();
    }
}
```

静态内部类只可以访问外部类的静态属性方法，创建静态内部类的方式也暗示了其不拥有对外部类对象的引用

```
public class OuterClass {
    private static String name = "sfas";
    static class InnerClass{
        public void test(){
            System.out.println(name);
        }
    }
    public static void main(String[] args) {
        InnerClass in = new InnerClass();
    }
}
```

1.1 .this 与.new

对于生成外部类对象的引用可以使用外部类名加.this 的形式。

如果需要在非静态方法之外的任意位置创建内部类对象就需要使用外部类对象.new

1.2 内部类的向上转型

private 内部类给类的设计者提供了一种途径，通过这种方式可以完全阻止任何依赖于类型的编码，并且完全隐藏了实现的细节。内部类（某个接口的实现或者基类的子类）能够完全不可见，并且不可用，所得到的只是指向基类或者接口的引用。

1.3 局部内部类（在一个方法中的类）

此局部内部类是方法的一部分，在方法之外是不能访问的，一旦方法执行完毕，局部内部类对象就不存在了。使用局部内部类而不使用匿名内部类的原因是需要不止一个该内部类的对象。

```
public class OuterClass {
    private static String name = "sfas";

    public void test() {
        class InnerClass{

        }
    }
}
```

1.4 匿名内部类

形如 new BasicObject(){}

创建一个没有名字的对象，BasicObject 可以是一个基类，那么创建的就是基类的匿名子类，大括号中是对基类的扩展，同样，BasicObject 也可以是一个接口，创建的是接口的一个实现，你可以用一个基类或者接口来接受匿名内部类的引用，这样匿名内部类就会自动被向上转型。

如果在匿名内部类中想要使用一个在其外部定义的对象时，那么这个引用必须是 final。如果想在匿名内部类想做一些类似构造器的行为，字段的赋值可以简单的传 final 就行，如果想做一些复杂的事情，则可以通过实例初始化的方法，如下：

```
public class OuterClass {
    private static String name = "sfas";

    public static void main(String[] args) {

        new Runnable() {
            {
                System.out.println("假装这是匿名内部类的构造方法");
            }

            public void run() {

            }
        };
    }
}
```

```
Problems @ Javadoc Declaration Console L
<terminated> OuterClass [Java Application] D:\Major progra
假装这是匿名内部类的构造方法
```

TIP：如果参数是传给基类构造器使用的话，由于它并不会在匿名内部类被直接使用，所以不需要 final。

1.5 静态内部类（嵌套类）

不需要外部类的对象，不能访问非静态的外部类方法属性。非静态内

部类不能有 static 属性或方法，而静态内部类可以有。

1.6 接口中的内部类

接口中的内部类是 public 和 static 的，接口的（接口的属性和方法都是 public）

11.7 内部类存在的必要

每个内部类都可以独立地继承一个接口的实现，无论外部类是否继承了某个内部类的实现，对于内部类来说都没有影响。内部类提供了可以继承多个具体或抽象类的能力，使多继承的解决方法变得完整。通过内部类我们可以很容易地实现一个接口的多种实现方式。

1.8 闭包

闭包是一个可调用的对象，它记录了一些信息，而这些信息来自于创建它的作用域。内部类是 java 中的闭包，它包含了外部类的信息，还拥有一个指向外部类对象的引用。

1.9 内部类的继承

内部类继承时不能使用默认构造方法，必须使用带有外部类的引用的构造方法，在内部类的子类构造方法中还必须调用外部类的构造方法，以确保继承子类拥有外部类的引用。

2. 集合类

List、Set、Queue、Map 基本类型

List 必须按照插入的顺序保存元素，Set 不能有重复的元素，Queue 按照排队规则来确定对象产生的顺序（通常与他们被插入的顺序相同）。Map 键值对形式存在。

2.1 添加一组元素

java.util 包中的 Arrays 和 Collections 类提供了很多实用的方法，可以在一个 Collection 中添加一组元素。Arrays.asList()可以接受一个数组或者逗号分隔的可变参数列表，并将其转化为一个 List 对象。Collections.addAll()方法可以接受一个 Collection 对象，以及一个数组或者用逗号分隔的可变参数列表，并将元素其加入到 Collection 中。Collection.addAll 只能接受另一个 Collection 对象作为参数。

2.2 存储顺序

ArrayList 和 LinkedList 都按照被插入的顺序保存元素。

Set 相同项只保留一个。HashSet 存储方法较复杂，但是可以最快获取元素。TreeSet 按照比较结果的升序存储。LinkedHashSet 按照插入的顺序保存。

HashMap、TreeMap、LinkedMap 特点与上相同。

2.3 Queue

队列遵循先进先出的原则。队列在并发编程中特别重要。

LinkedList 实现了 Queue 接口，因此 LinkedList 可以用作 Queue 的一种实现。将 LinkeList 向上转型为 Queue。offer()方法在允许的情况下将一个元素插入队尾，或者返回 false。peak()和 element()返回队头，并且不会移除元素，peak()在队头为空时返回 Null,而 element()则会抛出 NoSuchElementException 异常。poll()和 remove()方法移除并返回队头，poll()在队头为空时返回 Null,remove()则抛出异常。

PriorityQueue：

当在 PriorityQueue 上调用 offer()方法来插入一个对象时，这个对象会在队列中被排序。默认排序使用对象在队列中的自然顺序，可以通过提供自己的 Comparator 来修改这个顺序。Priority 确保你调用上述方法时，获得的是优先级最高的元素。Integer,String,Character 可以与 PriorityQueue 一起工作，因为这些类已经内建了自然排序。如果想使用自己的类，就必须包括额外的功能以产生自然排序或者提供自己的 Comparator。

3. 反射机制

反射允许我们在运行时发现和使用类的信息。

3.1 Class 对象

每个类都有一个 Class 对象，每当编译并且编译了一个新类，就会产

生一个 Class 对象（更恰当的说，是被保存在一个同名的.class 文件中）。运行程序的 Java 虚拟机（JVM）使用“类加载器”子系统生成这个类的对象。

通过 Class.forName(“类的全称”)或者类对象的 getClass()方法获取 Class 引用

3.2 isInstance()，instanceof()

判断是否是该类或者该类基类的实例

3.3 java.lang.reflect

getFilelds(),getMethods 和 getConstructors(),返回属性，方法，构造器的对象数组。用 invoke()方法调用与 Method 对象关联的方法。

4. I/O 系统

4.1 File 类

可以代表一个特定文件的名称，有能代表一个目录下一组文件的名称。

```
2
3  import java.io.File;
4  import com.superui.filefilter.DirFilter;
5
6  public class DirList {
7      public static void main(String[] args) {
8          File path = new File("H:/");
9          String[] list = path.list(new DirFilter("^p.*"));
10         for (String string : list) {
11             System.out.println(string);
12         }
13     }
14 }
15
```

FilenameFilter实例，此list方法会为此目录对象下的为一个文件调用accept方法

目录过滤器：

```java
public class DirFilter implements FilenameFilter {
    private Pattern pattern;

    public DirFilter(String regex) {
        pattern =Pattern.compile(regex);
    }

    @Override
    public boolean accept(File dir, String name) {
        return pattern.matcher(name).matches();
    }
}
```

过滤方法

File 类不仅仅只代表存在的文件或目录，也可以用 File 对象来创建新的目录或尚不存在的整个目录路径。还可以查看文件的特性（大小，最后修改日期，读/写），检查某个 File 对象代表的是一个文件还是一个目录，并且还可以删除文件。

方法介绍：

File f = new File(“filepath”);

f.isFile()

f.isDirectory()

f.mkdirs()

f.renameTo()把文件重命名或移动到完全不同新路径

4.2 输入与输出

InputStream 用来表示那些从不同数据源产生字节输入的类，包括：

（1）字节数组

（2）String 对象

（3）文件

（4）“管道”，一端输入，另一端输出