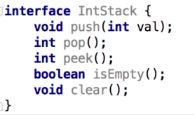
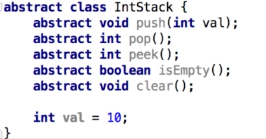
Interface

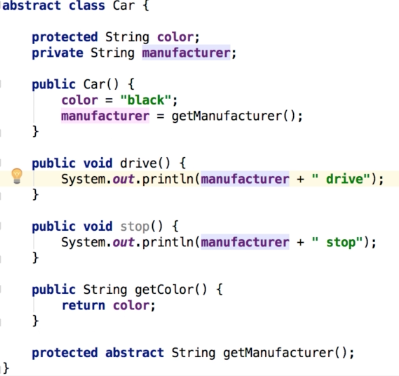


1. 可以被视为是规则 由类来遵守
2. 接口里声明的变量都是常量，不可更改
3. 接口里的函数只需要定义 不需要实现，实现交给类
4. 方法和常量不需要写任何修饰 都默认是public
5. 类必须实现所有的方法// 在IDE用option+alt+回车键 出现implement methods 可以直接将所有的方法生成
6. @override 是给程序员看的，一眼知道这个方法来自于interface或者类
7. Java8 引入了default method—>可以在接口里实现的方法，这样接口和抽象类的区别就更小了：
   * 单继承 多实现
   * 抽象类可以有protected和private类型的变量和方法，接口不可以
8. 为什么要使用interface？
   * ArrayList和LinkedList都是实现了List接口的类，如果有两个方法，他们除了参数一个是ArrayList另一个是LinkedList之外没有任何区别，如果使用接口，就可以简化成同一个方法，便于程序的的维护。
     + Int sumList(ArrayList<Integer> list)
     + Int sumList(LinkedList<Integer> list)
     + Int sumList(List<Integer> list)
9. 当将一个类的对象的reference 赋予一个接口的时候，这个类特有的方法和变量就不能够使用了。
   * List<Integer> thislist=new LinkedList<>();
   * Thislist.addFirst(); error

Abstract Class



1. 在一定程度上可以和interface互换
2. C++没有interface，只有class，是多继承
3. Java 类是单继承，但是可以实现很多接口
4. 抽象方法只能在抽象类中定义
5. 为什么使用抽象类？
   1. 抽象类不能够直接new出来对象，因为没啥意义
   2. 如果子类要重写父类的一些方法，那么父类自身实现这些方法就没有意义了。就直接将方法变成抽象方法，不去实现它。
6. 抽象类可以有不抽象的部分，可以实现一些非抽象类。
7. 抽象方法要么是protected要么是public 因为抽象方法不被访问是没有意义的



1. 抽象类演化的结果：
   1. 将一些方法实现，就不需要每一个子类重写了。
   2. 如果没有最后一个抽象方法，那么子类如果忘记给manufacturer赋值，调用drive等方法就会变成null drive。为了避免这一错误，在抽象类里定义一个抽象类，强制你在子类里必须实现给manufacturer赋值这个操作。同时manufacturer在抽象类里就可以变成private，因为子类不需要访问。
   3. 注意构造函数的改变。