第一部分

与参考设计类似的部分

1. 参考设计的Canvas类在PJ的设计中功能类似的是View类
   类似于strategy设计模式
   优点：减少了判断语句的使用，在创建类的时候直接确定具体策略。
2. 将text,line等指令设计成类
   并且方法也是类似的，在参考设计里是draw(offsetPoint, canvas)，在PJ实现里是execute(View*)
   优点：利于封装，也利于后期的undo/redo操作。
   可能的缺点：在统一继承基类的基础上可读性以及复杂程度未必优于设计成函数的情况。

第二部分

与参考设计不同的部分

1. 关于不同的坐标系与颜色
   参考设计中将Device2与Device256分开；将两种坐标系利用strategy模式继承抽象类CoordinateSystem，通过Device进行操作；PJ实现使用Coord抽象类，Coord_mono与Coord_gray为派生类进行画布的构造。
   PJ实现的缺点：仍然存在很多地方需要if...else...判断，开销较大，未能将坐标系抽象出来。优点在于阅读较为容易。
   参考设计的结构更加清晰明确，代码重用度低，并且开销较小。
2. 关于统计信息
   参考设计中使用observer设计模式，直接使用Canvas类组合CanvasEventsObserver，传递数据。
   PJ实现中通过ScriptProcessor组合MacroVec类进行数据统计。（主要是未能很好地想到如何通过View类直接进行统计）
   PJ实现的缺点：在ScriptProcessor中实现了过多的方法以及组合过多的类，在使用指针的时候调用层次过多，开销比较大。以及MacroVec的封装并不好，这里对于STL的封装有局限性。
   可能可以考虑在命令里组合observer实现数据统计，能够比较明确。
3. 关于宏（组合命令）
   参考设计里是用组合模式进行处理 leaf为各个指令而view作为Composite；PJ实现里没有使用composite模式而是直接在宏定义的时候创建View对象，并且在MacroVec里面记录。最后使用的时候用showViewAt在主画布上显示。
   PJ实现的缺点，没有很好地抽象出组合命令的结构，导致在ScriptProcessor中比较混乱。
   参考设计可能存在的问题：当加上command模式的时候会遇到问题，可能会需要继承多个类以及复杂的方法。

第三部分

利用设计决策的优点

1. 使用command设计模式
   主要使用Command类作为抽象类；ConcreteCommand文件中的类继承了command为各个命令的具体实现；CommandExecutor类用于控制命令并且记录；这里使用stack对命令进行存储；ScriptProcessor类用作读取脚本命令并且将命令传递给CommandExecutor，相当于（Invoker）的作用。

在实现undo/redo方面较好的使用的command模式的特性，将每个命令设计成类，并且将其放到两个栈（UndoStack, RedoStack）里面，当需要undo/redo的时候进行变换；没有使用保存每次全部画布的办法，而是给每个类设计了不同的undo/redo方法，节省了内存。在color中利用static的成员stack实现undo/redo，因为这样可以使得不同的color类能够共享同样的栈。