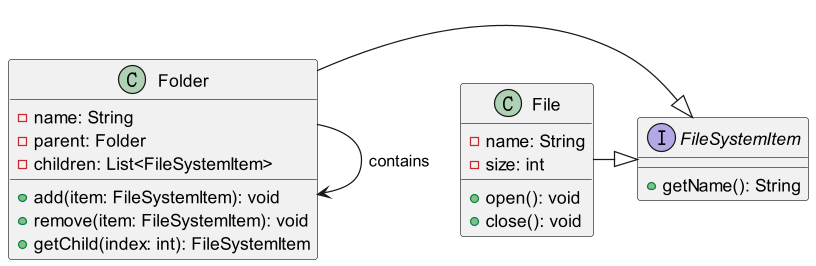
1. 如图所示的司机驾车设计方案存在什么问题 ? **（4 分）**进一步阐述如何评价“组 织良好的”设计类 ？**（6 分）**

**违反单一职责原则。**“司机”类中包含了“驾驶车辆”的行为如 go() 和 stop() 方法。司机不直接控制这些动作，而是通过车辆来实现。**耦合性高。**司机类与车辆类的关系过于直接，导致两者之间强耦合。当需要更改车辆的实现，例如添加新的功能或使用其他车辆类型时，可能需要同时修改司机类。**扩展性差。**如果需要支持其他类型的交通工具例如自行车、摩托车等，现有设计缺乏灵活性。

一个好的设计类，应该遵循单一职责原则。低耦合高内聚、可扩展性、符合现实语义、设计模式的使用、清晰的接口。

1. windows的资源管理器如图所示，若由你来系统实现，你将如何设计？请用类图来给出设计方案（7分），并阐述方案选择的理由（3分）？



模块化设计。FileSystemItem作为一个接口，定义了文件和文件夹的共有行为，符合开闭原则，便于扩展其他类型的文件系统项。

复合关系的体现。Folder类包含children列表，用于存储其下的所有文件和子文件夹，实现了树状结构的组织形式，贴合文件系统的真实情况。

将文件和文件夹的职责分开，File类专注于文件相关操作（如open和close），Folder类负责管理子节点。符合单一职责原则。

灵活性与扩展性。设计支持未来扩展，例如增加其他类型的FileSystemItem（如快捷方式、压缩文件等），只需实现FileSystemItem接口即可，系统不需要大幅修改。

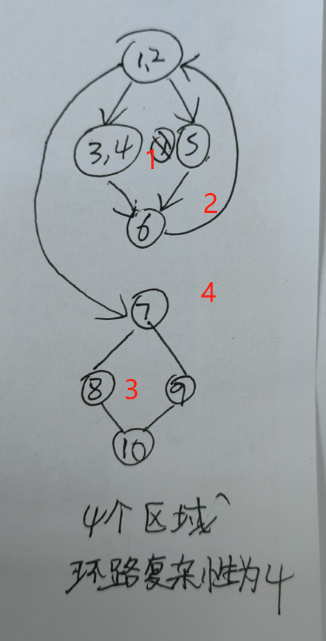
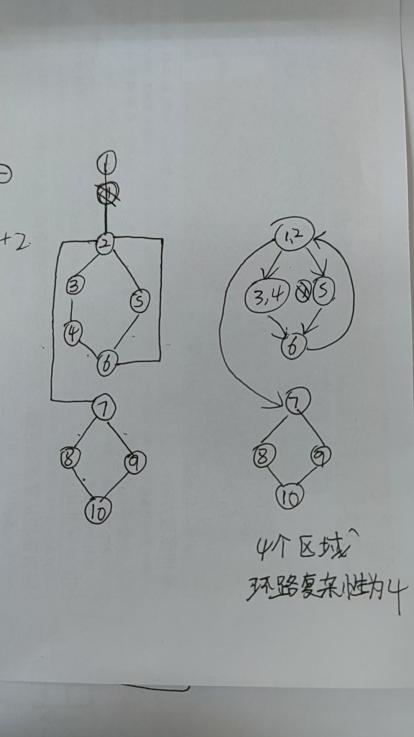
现实语义映射。类与现实世界的文件系统概念相符，易于理解和维护。例如，Folder与File分别对应文件夹和文件，用户和开发者都容易识别。

3、如图所示，在家庭安保 SafeHome 系统设计中，图（a）给出了系统安全功能 的顺序图，图（b）给出了 ControlPanel 类的状态图，请解释两种建模方式的区别是什么？（10 分）

顺序图描述系统中不同对象之间的交互顺序，以及消息的发送和接收过程。它主要关注消息的**时间顺序**。强调对象之间的动态交互，展现了行为的时间轴。当需要清晰表达系统的操作流程和对象间的调用顺序时，使用顺序图更为合适。表现的是方法调用的时间流向。有助于理解复杂的系统操作流程。例如图 (a) 中，从 Homeowner 到 Control Panel 再到 System 和 Sensors，清晰地展示了各个步骤的执行顺序。

状态图描述单个对象在生命周期中的状态变化，以及引起这些变化的事件。强调对象从一种状态转换到另一种状态的条件和触发因素。当需要展示某个对象的所有可能状态及其转换条件时，状态图更为合适。以状态为中心，而不是方法调用。有助于理解对象行为的完整生命周期。例如图b中，Control Panel 的状态变化从 Reading 到 Comparing，再到 Locked 和 Selecting，展示了其在整个生命周期中的可能状态。

4、



路径

1-2-3-4-6-1-2-7-8-10

1-2-3-4-6-1-2-7-9-10

1-2-5-6-1-2-7-8-10

1-2-5-6-1-2-7-9-10

II

1、项目开始时，为了导出需求，向客户和其他利益相关者提问了如下 3 个问题，请

尝试分析每个问题的目标是什么（9 分），对系统构建的方案选择有什么作用（6 分）？

1) 成功的解决方案将带来什么样的经济效益 ？

2) 如何描述由成功的解决方案产生的“良好”输出的特征 ？

3) 你是回答问题的合适人选吗 ？

第一个问题的目标集中于客户和其他利益相关者以及整体目标和收益。第二个问题有助于软件开发组更好地理解问题，并允许客户表达其对解决方案的看法。第三个问题关注沟通活动本身的效率。

上面的问题有助于**明确需求优先级。**通过上述问题，开发团队能够明确客户真正关心的经济效益和功能目标，帮助优先选择能最大化实现客户期望的解决方案。**降低开发风险。**识别合适的利益相关者和明确需求标准能够避免因需求不准确或需求变化造成的方案调整风险，从而选择更稳定、可行的方案。**增强方案适配性。**通过衡量经济效益和输出特征，方案的设计能更好地适应客户的业务需求，确保最终系统的功能性和效益性达成一致。**提高验收成功率。**通过定义清晰的“良好”输出特征，团队可以确保系统在交付时满足客户验收标准，从而提高项目成功率。

2、软件工程的实过程施框架定了 5 个框架活动与 8 个普适性活动，每一项活动都有其专有的动作、任务和内涵。但软件工程不应该是教条的法则，不要求项目团队机械执行，可以根据项目要解决的问题、特点与团队文化等因素，进行适应性调整，请根据自己的理解，简单描述每一个框架活动的任务（6 分），并谈一下不同项目的实施过程，应从哪些方面来调整（9 分）。

**沟通。**通过与客户、利益相关者以及团队成员的沟通，明确系统需求和功能目标，确保双方对需求的理解一致。形成清晰的需求文档和产品规格说明书，作为开发的基础。**策划。**制定项目计划，包括任务分解、资源分配、时间表、风险评估和成本估算。为项目实施提供明确的方向，合理分配时间和资源，避免项目脱轨。**建模。**通过建模工具分析和设计系统结构，定义系统功能模块和接口。为开发阶段提供蓝图和技术指导，确保设计的合理性和功能可实现性。**构建。**完成系统的编码、集成和测试，最终实现系统的功能目标。生成满足需求的可运行软件系统。**部署。**将开发完成的软件交付用户，包括安装、配置、用户培训，以及收集用户反馈以进行改进。确保软件顺利上线并满足用户需求。

**软件项目跟踪与控制。**小型项目使用轻量工具（如看板）；大型项目设置详细里程碑。

确保项目按计划推进，及时发现问题。**风险管理。**高风险项目制定详细的缓解计划；低风险项目聚焦关键风险。减少风险对项目的影响。**软件质量保证。**技术复杂项目加强自动化测试；简单项目减少审查频率。提高系统稳定性，减少缺陷。**技术评审。**经验丰富团队减少正式评审；复杂项目增加评审频率。发现设计问题，提高开发质量。**测量。**时间紧项目聚焦关键指标；长期项目跟踪全面指标。优化过程，支持数据决策。**软件配置管理。**快速项目使用自动化工具；稳定需求项目注重文档完整性。防止版本混乱，管理变更。**可复用管理。**复用成熟团队资源；新项目积累可复用模块。减少重复开发，节约成本。**工作产品准备与生产。**内部项目简化文档；外部项目完善用户手册和交付品。满足用户需求，提升满意度。

III

1、1963～1966 年，开发的 IBM 360 机操作系统共 4000 多个模块，约 100 万条指令，投入 5000 人年，耗资数亿美元，结果延期交付后系统中仍发现大量（2000个以上）的错误，项目负责人 F.D.Brooks 总结教训时提到：问题主要原因是管理出了问题，团队构建发生了问题。**请根据所学知识，简析选择团队机构应考虑什么因素 （7 分）？并谈一下个人对构建团队的认识（8 分）**

**团队规模。**规模过大会增加沟通成本，规模过小会影响工作效率。**团队成员的技能匹配。**确保团队成员具备完成任务所需的技能，包括技术能力、项目管理经验等。**沟通机制。**确保团队成员能够顺畅交流，减少信息传递中的误解。**角色分配与责任明确。**每个团队成员的职责和任务要清晰，避免出现责任不明或任务重叠的情况。**团队文化与协作。**团队成员应有协作精神。**项目复杂性。**根据项目的规模和复杂性，合理配置团队结构。**管理方式。**项目管理方式，如敏捷或瀑布模型。应与团队特点和项目需求匹配。

**团队构建是项目成功的关键。**团队的结构、规模、能力和协作方式直接影响项目的效率和最终交付质量。一个合适的团队能够更高效地解决复杂问题。**团队规模需要合理。**团队规模应与项目复杂度和工作量匹配。过大可能导致沟通成本增加，过小则可能因资源不足而影响进度。**团队成员的技能匹配很重要。**确保每个团队成员的技能能够满足其职责的需求。例如，技术开发需要技术专家，管理岗位需要具备协调能力的人员。**清晰的分工与职责划分。**团队构建过程中需要明确每个成员的职责，避免责任重叠或模糊，确保每个模块有专人负责。**建立高效的沟通机制。**沟通是团队协作的核心。需要在团队中构建高效的沟通机制，减少信息传递中的失真或延迟。**关注团队文化与协作氛围。**团队成员之间需要信任和协作精神。良好的团队文化能够增强团队凝聚力，提升团队解决问题的能力。**动态调整团队结构。**项目需求可能随着时间发生变化，团队结构需要能够灵活调整。例如，在不同阶段增加测试人员或开发人员。**团队构建需要以项目目标为核心。**团队的所有构建工作都应该围绕项目目标进行，避免因团队管理问题偏离项目核心目标。

1. 一款游戏软件的局部需求描述如下：
2. 一个游戏人物有头、身体与胳膊、腿等等四肢组合而成。

2) 为实现游戏人物动作的协调一致，设计人员给出了如图所示的设计方案。

**对此方案，体现了哪些面向对象的基本概念？（6分）尝试探析设计是如何运用面向对象设计的依赖倒置原则、LisKov 替换原则，来满足面向对象的开闭原则的 （9分）。**

类。图中定义了Player类以及其组成部分（PlayerBody、PlayerArms、PlayerLegs、PlayerHead），每个类都封装了自身的属性和方法。

继承。Player类实现了GameRule接口，继承了接口中的行为定义（如update()和display()方法），这体现了代码的复用性和灵活扩展性。

封装。每个类都有自己的属性（如position、orientation、color、speed）和行为（如update()和display()），实现了数据与方法的封装。

多态。Player类通过实现GameRule接口，可以多态地调用update()和display()方法。

组合。玩家（Player）是由多个部分（如PlayerBody、PlayerArms等）组合而成的，这体现了整体与部分的组合关系。

接口。GameRule接口定义了行为规范，Player类通过实现接口，保证了系统模块之间的行为一致性。

依赖倒置原则。Player类依赖于GameRule接口，而不是具体实现。通过接口设计，确保了高层（游戏规则）与低层（玩家实现）之间的依赖关系稳定且灵活。

Liskov 替换原则。Player类实现了GameRule接口，任何地方如果需要GameRule类型的对象，都可以用Player实例替换。此外，Player的各个组成部分（如PlayerBody、PlayerArms等）也遵循相同的属性与方法设计，保持了一致性。

开闭原则。Player类的设计通过组合方式（如PlayerBody、PlayerArms等）实现了灵活扩展，不需要修改Player类即可添加或调整新的身体部分。实现GameRule接口的方式也允许在无需修改接口的情况下扩展新的实现。