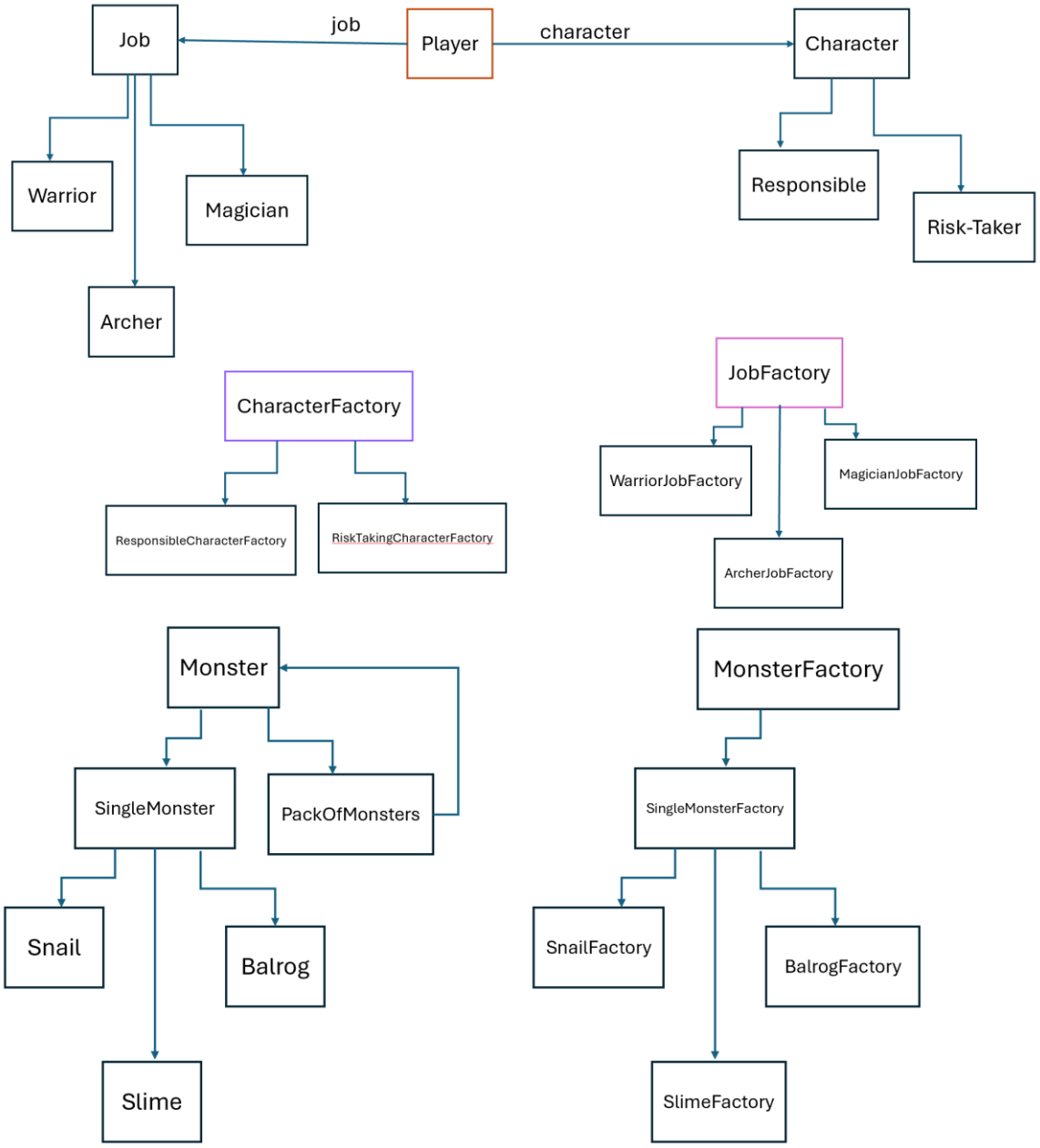
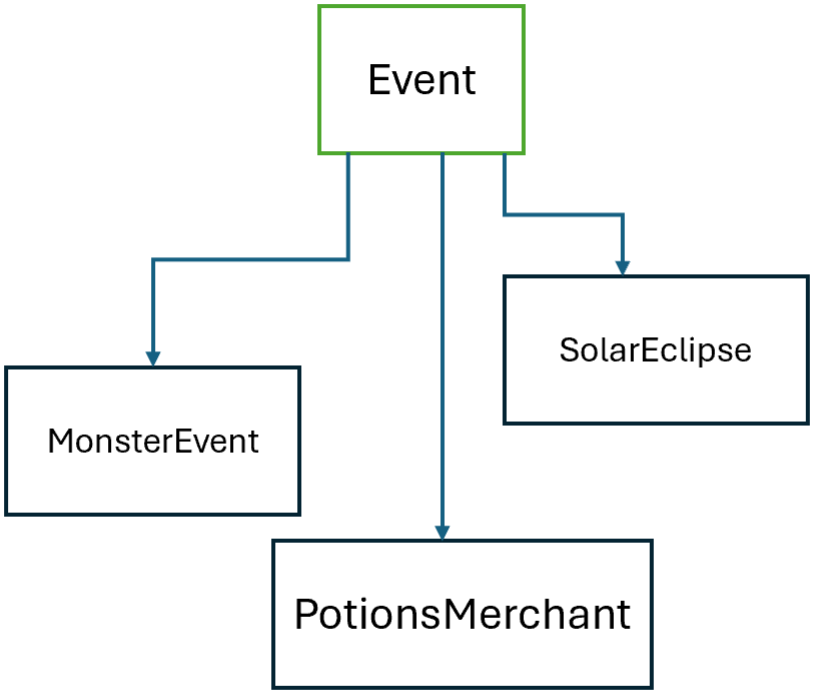
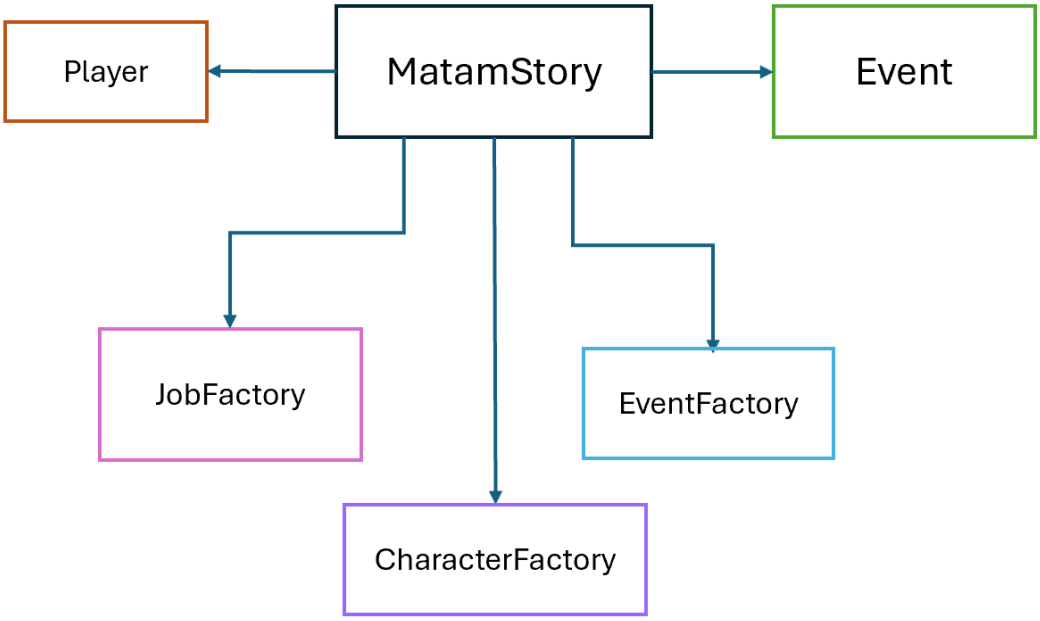
חלק יבש – שאלה 1 – UML:




Event
MonsterEvent
SolarEclipse
PotionsMerchant


Player
MatamStory
Event
JobFactory
EventFactory
CharacterFactory

שאלה 2 – design patterns:

כמובן שהשתמשנו בdesign patterns:

1. Abstract factory – השמשנו עבור Job (כאשר JobFactory הוא האבסטרקטי ו WarriorFactory, Archer Factory וכו' יורשים ממנו וממשים את הפונק' של יצירת Job כזה או אחר בהתאם למחלקה (למשל warriorFactory יחזיר job מסוג warrior (שיורש מjob)).

2. Abstract factory – השתמשנו עבור character (כאשר CharacterFactory הוא האבסטרקטי וRiskTakingFactory, Responsible Factory וכו' יורשים ממנו וממשים את הפונק' של יצירת Character כזה או אחר בהתאם למחלקה (למשל responsibleFactory יחזיר charcter מסוג Responsible (שיורש מ character)).

3. Composite – רצינו שמפלצת תוכל להיות גם מפלצת יחידה (למשל חילזון) או להקה של מפלצות, אבל שגם להקה של מפלצות תוכל להכי בתוכה איבר מסוג להקה (=תת להקה). לכן מימשנו את Monster בעזרת composite: Monster היא מחלקת אב אבסטרקטית שרק "מכתיבה" אילו פונק' הילדים יצטרכו לממש. SingleMonstera יורשת ממנה וגם היא אבסטרקטית בחלקה (ממשת פונק' גנרית של כל מפלצת בודדת), וממנה יורשות Slime, Balrog,Snail שלכל אחת ctor משלה (בהתאם לערכיה) ופונק' מתאימות (למשל לbalrog פונק' התמודדות עם שחקן שמעלה לה את הכוח ב2).

   ויש את MonsterPack שמכילה וקטור של מפלצות, וממשת כל פונק' בתורה כסכימה של המפלצות שלה (היומי הוא שלכל מפלצת, גם לתת להקה, יש את הממשק האחיד, שמכיל למשל getCombatPower כסכום של כל הכוחות של כלל המפלצות בלהקה).

4. Strategy + command – השתמשנו בdesign הזה לניהול המשחק – במקום שלכל שחקן יהיה למשל שדה string שמכיל את שם ה job ובכל התמודדות נבדוק מה הערך, ונבצע פונקציונליות אחרת, וכמובן נבדוק את סוג האירוע (ואז יהיו הרבה תנאים מקוננים ולוגיקה מסובכת) לכל אירוע יש מתודת handle Event משלה שמקבלת שחקן וקוראת לפונק' "ההתמודדות" שלה. למשל SolarEclipse::handleEvent יקרא לplayer.reactToSolarEclipse. וplayer בתוכוף, במקום לבדוק מה הjob שלי, יכיל אובייקט מסוג job וCharacter ויקרא לjob.handleMonster וכדומה, והפונק' הרלוונטית למקצוע הספציפית תתרחש בתוך warrior.handleMonster למשל. האנקפסולציה של player.react בתוך event.handle מהווה את המימוש של הcommad (ביחד עם strategy) והמימוש של job/character כאובייקט חבר מחלקה של player מהווה חלק מהמימוש של strategy.

שאלה 3 – הרחבת job:

הודות לstrategy שמימשנו, ההוספה של rogue תהיה די פשוטה. נממש מחלקה חדשה בשם rogue שתירש מjob ותממש בתוכה את הפונק' הרלוונטית (למשל אם היא לא מגיבה באופן שונה לליקוי חמה, נשאיר את הפונק' במחלקת האב של Job) ועבור התמודדות עם מפלצת, נממש את combatMonster (שנמצאת בממשק של job כפי שנרצה – נבדוק אם הכוח גדול פי2 – אם כן "נתחמק", אחרת נילחם כרגיל).

בנוסף נוסיף rogueFactory שירש מJobFactory ויחזיר job מסוג גנב.

בקלט מהקובץ, כל פעם שניתקל במקצוע מסוג גנב, נקרא לפונק' היצירה בfactory של rogue (בתוך פונק' גנרית קיימת שמקבלת שם job ומחזירה את הפקטורי המתאים).

שאלה 4 – הרחבת event:

שוב, הודות לdesign patterns, לא נדרשת הרבה עבודה כדי להוסיף אירוע חדש. נוסיף מחלקה בשם divineInspiration שיורשת מevent וממשת handleEvent שבתוכה תקרא לplayer.handleDivineInspiration (נוסיף פונק' כזאת לPlayer). ובפונק' player.handleDivineInspiration נייצר job חדש באופן רנדומלי (באמצעות הfactories שיש לנו לכל מקצוע, ונשים את המצביע למקצוע החדש בתוך player.job (זהו data member).

מימ לתחרות מימז: