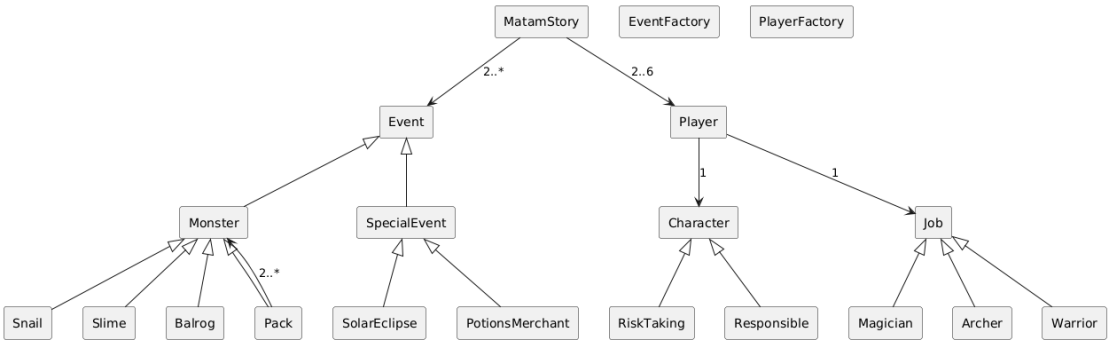
1. להלן דיאגרמת המחלקות:



2. להלן תבניות התכן שהשתמשנו בהם במהלך פתרון התרגיל ומיקומם :

- "Composite":
  עבור "Monster" ו – "Pack". כפי שניתן לראות "Pack" מכיל בתוכו לפחות 2 "Monster" שהוא מחלקת האב של "Pack"

- "Strategy":
  ב – Player הואיל וניתן לשנות את Job ואת Character בזמן ריצה אם נרצה בכך.

- "Command":
  ב – Event כך שלכל Event יצרנו Play משלו ויהיה ניתן להוסיף לו בעת הצורך undo או עוד פעולות אם נרצה בכך.

- "Singelton":
  השתמשנו בתכן Singelton עבור יצירת PlayerFactory ו – EventFactory, דבר זה נצרך הואיל ונרצה שאותו פקטורי ייווצר פעם אחת בדיוק, יחיה לאורך כל קיום התוכנית ויהיה נגיש מכל מקום, לצורך בניית עצמים והפעלת מתודות של המחלקות.

3. עקב בנית הקוד בצורה מודולרית תוך כדי שימוש ב PlayerFactory בצורה שבה קל יותר להוסיף Character ו - Job חדשים עם מעט מאוד שינוי לקוד הקיים.

נוסיף תחילה קובץ Rogue.h את המחלקה היורשת Rough בצורה הבאה:

```
class Rogue : public Job {

public:
    Rogue();
};
```

ואת הבנאי בקובץ Rogue.cpp בצורה הבאה:

```
Rogue::Rogue() : Job("Rogue") {
    m_combatQuality = "Evasion";
}
```

לאחר מכן נוסיף לקובץ PlayerFactory.cpp את השורה הבאה בשביל שנוכל ליצר אוביקט מסוג Rogue:

```
PlayerFactory::PlayerFactory() {
    // Register Default Job Creators:
    m_jobCreators["Warrior"] = [] { return make_unique<Warrior>(); };
    m_jobCreators["Archer"] = [] { return make_unique<Archer>(); };
    m_jobCreators["Magician"] = [] { return make_unique<Magician>(); };
    m_jobCreators["Rogue"] = [] { return make_unique<Rogue>(); };

    // Register Default Character Creators:
    m_characterCreators["Responsible"] = [] { return make_unique<Responsible>(); };
    m_characterCreators["RiskTaking"] = [] { return make_unique<RiskTaking>(); };
}
```

ונוסיף את מתודת evasionPlay עבורו בקובץ Monster.cpp בצורה הבאה ואת ההכרזה בקובץ ההדר כדי שנוכל להשתמש בו במתודה Play של Event:

```
string Monster::evasionPlay(Player& player) const {
    if (m_combatPower >= ROUGH_FORCE_DIFFERANCE_CONDITION_VALUE * player.getForce()) {
        return player.getName() + " dodged the encunter!";
    }
    return distancePlay(player);
}
```

נוסיף רישום של המתודה החדשה ל EventFactory::PlayMethod בקובץ Monster.cpp בצורה הבאה:

```
Monster::Monster(string name, int combatPower, int loot, int damage)
    : m_monsterName(move(name)), m_combatPower(combatPower), m_loot(loot), m_damage(damage) {
    EventFactory::getInstance().registerPlayMethod(m_monsterName + "_" + "Melee",
        [this](Player& player) { return meleePlay(player); });
    EventFactory::getInstance().registerPlayMethod(m_monsterName + "_" + "Distance",
        [this](Player& player) { return distancePlay(player); });
    EventFactory::getInstance().registerPlayMethod(m_monsterName + "_" + "Evasion",
        [this](Player& player) { return evasionPlay(player); });
}
```

ניתן לראות כי אכן המערכת תפעל כשורה ולא נצטרך לשנות את המימוש רק להוסיף קטעי קוד במקומות הרלוונטים.

4. ניתן לראות כי גם במקרה בו נרצה להוסיף Event מסוג DivineInspiration עקב המודולריות השינויים יהיו מזעריים, ובנוסף היכולת של Player לשנות את Job עקב שימוש ב - Strategy במימוש של Player השינויים יראו כך:

נוסיף לקובץ SpecialEvent.h את המחלקה בצורה הבאה:

```
class DivineInspiration : public SpecialEvent {

public:
    DivineInspiration();
    string play(Player& player) override;
};
```

ונוסיף גם לקובץ SpecialEvent.cpp את הבנאי והמתודה Play בצורה הבאה:

```
DivineInspiration ::DivineInspiration () : SpecialEvent("DivineInspiration ") {}

//*****************************************************************************//

string DivineInspiration ::play(Player &player)
{
    // For exemple: change to Warrior. needs to define a method for instanciate random option.
    // Ofcourse, it is possiable that the new random job needs to be differante from the old one
    player.setJob("Warrior");
    return player.getName() + " had a Devine Inspiration and changed his job to Warrior";
}
```

ובקובץ EventFactory.cpp נוסיף את השורה הבאה שנוכל ליצור אובייקט Event מסוג DivineInspiration:

```
EventFactory::EventFactory() {
    // Register Default Event Creators:
    m_eventCreators["Snail"] = [] { return make_unique<Snail>(); };
    m_eventCreators["Slime"] = [] { return make_unique<Slime>(); };
    m_eventCreators["Balrog"] = [] { return make_unique<Balrog>(); };
    m_eventCreators["Pack"] = [] { return make_unique<Pack>(); };
    m_eventCreators["SolarEclipse"] = [] { return make_unique<SolarEclipse>(); };
    m_eventCreators["PotionsMerchant"] = [] { return make_unique<PotionsMerchant>(); };
    m_eventCreators["DivineInspiration"] = [] { return make_unique<DivineInspiration>(); };
}
```

וגם כאן ניתן לראות כי אכן המערכת תפעל כשורה ולא נצטרך לשנות את המימוש רק להוסיף קטעי קוד במקומות הרלוונטים.