Bitte verwenden Sie für Ihre Lösungen einen ordentlichen Programmierstil. Vor allem auf eine korrekte Formatierung, aussagekräftige Variablennamen und ausführliche Kommentare sollte dabei Wert gelegt werden.

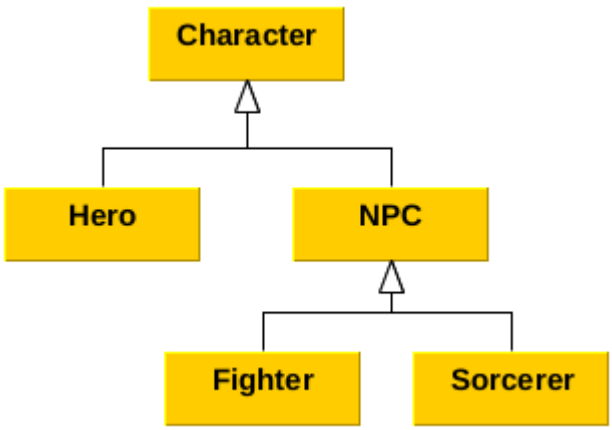
Methoden der Spieleentwicklung III (100%)

Aufgrund der Tatsache, dass die letzte Version des Codes sehr viele Duplikate bzw. Kopien von Funktionen enthielt, müssen Änderungen an der Programmlogik auch mehrmals durchgeführt werden. Aus diesem Grund haben Sie beschlossen, die erlernten Konzepte der **Vererbung/Generalisierung** und des **Polymorphismus** in der nächsten Version des Codes umzusetzen.

Des Weiteren wollen Sie Fehlerquellen möglichst minimieren, damit es zu weniger Bugs bei der weiteren Entwicklung kommt. Daher sollen die init-Funktionen durch entsprechende **Konstruktoren** ersetzt werden.

Daraus ergeben sich folgende Änderungen:

- Für die Heldin und die Feinde führen wir eine Verbungshierarchie ein (siehe Klassendiagramm unten). Als oberste Basisklasse dient nun die Klasse `Character`, die alle Eigenschaften und alles Verhalten beinhaltet, die von den beiden Unterklassen `Hero` und `NPC` (und deren Unterklassen) gemeinsam verwendet werden. Die Klasse `NPC` stellt die Basisklasse für alle Feinde dar.



- Die Klasse `Character` bekommt zwei zusätzliche Eigenschaften `Armor` (deutsch: Rüstungsstärke) und `Magic Resistance` (deutsch: Zauberresistenz), die Einfluss auf den Schaden nehmen, den ein Angriff auf einen Charakter verursacht. Die entsprechenden Anfangswerte sollen dem Konstruktor übergeben werden.
- In der nächsten Version Ihres Spiels soll auch die Diversität der Feinde erhöht werden. Deswegen sollen zwei neue Arten von Feinden hinzugefügt werden: Ein `Sorcerer` (deutsch: Zauberer) und ein `Fighter` (deutsch: Kämpfer).
    - Der Zauberer besitzt eine zusätzliche Eigenschaft, die `Magic Power` (deutsch: Zauberkraft).

    - Der Kämpfer besitzt auch eine zusätzliche Eigenschaft, die `Strength` (deutsch: Muskelkraft).

- Für die `attack()`-Funktion (die als gemeinsam genutzte Funktion in die Klasse `Character` gewandert ist) soll dynamische Bindung verwendet werden. Unterklassen überschreiben diese Funktion, um eine individuelle Schadensberechnung zu implementieren.
    - Die Heldin richtet einen Schaden gemäß folgender Formel an: `damage = rand(15, 25) - Armor`.
    - Der Zauberer richtet einen Schaden gemäß folgender Formel an: `damage = rand(5, 10) + MagicPower - MagicResistance`.
    - Der Kämpfer richtet einen Schaden gemäß folgender Formel an: `damage = rand(5, 10) + Strength - Armor`.
    - `rand(x, y)` steht dabei für eine Zufallszahl zwischen `x` und `y`.
    - `Armor` bzw. `MagicResistance` bezieht sich dabei auf die Werte des Gegners.
    - `Strength` bzw. `MagicPower` bezieht sich dabei auf die eigenen Werte.
- Wenn die Heldin einen Feind besiegt hat, soll ja ein zufällig ausgewählter Gegenstand aus dem Inventar des Feindes in das Inventar der Heldin übertragen werden. Um die weitere Entwicklung zu vereinfachen soll diese Funktionalität in der Objektfunktion `Item retrieveRandomLoot()` gekapselt werden, die einen zufälligen Gegenstand aus dem Inventar entfernt und zurückgibt. Überlegen Sie sich selbstständig, in welche Klasse diese Objektfunktion am besten hineinpasst.
- Wandeln Sie alle bisherigen init-Funktionen in Konstruktoren um. Fügen Sie weitere Konstruktoren hinzu, die Ihnen passend erscheinen.
- Implementieren Sie Destruktoren für die Klassen `Hero` und `NPC` hinzu, die eine beliebige Abschiedsmeldung ausgibt (z.B. "Heldin Annina verabschiedet sich und reitet in den Sonnenuntergang").
- Überladen Sie für alle Klassen den Output-Stream Operator `<<` mit für Sie sinnvoll erscheinenden Ausgaben.
- Speichern Sie wieder jede Klasse in eigenen Quelldateien (hero.{h, cpp}, character.{h, cpp}, npc.{h, cpp}, sorcerer.{h, cpp}, fighter.{h, cpp} und item.{h, cpp}). Die main-Funktion sollte in der Datei main.cpp zu finden sein.

Implementieren Sie wieder in der `main.cpp` ein Testprogramm, in dem die Heldin gegen zwei Feinde kämpft, den Zauberer Pascal und den Kämpfer Matthias. Verwenden Sie dabei für alle Ausgaben auf der Konsole den überladenen Stream-Operator `<<`, d.h. übergeben Sie die Objekte direkt dem Ausgabestream.

> **Info:** Sollten Sie keine Lösung für Exercise Sheet 2 haben, oder sicher sein wollen, dass Sie eine korrekte Basis für Exercise Sheet 3 verwenden, verwenden Sie bitte die