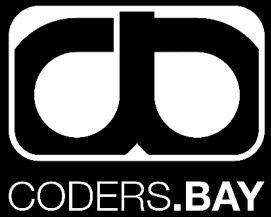
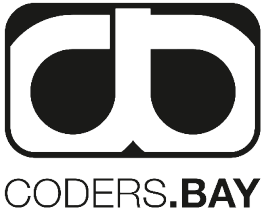


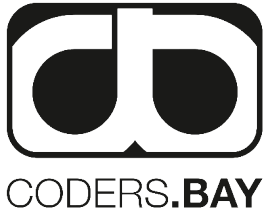

# Vererbung

# Beispiel RPG Game




In unserem RPG Game soll es eine Auswahl an verschiedenen Charakteren geben. Unter anderem gibt es **Zauberer, Elfen** und **Krieger**.
Sie alle haben einige für einen RPG Charakter wichtige Eigenschaften wie einen Namen und ein Geschlecht.
Zusätzlich haben sie unterschiedliche Fähigkeiten - ein Zauberer kann zaubern, ein Elf mit Pfeilen schießen und ein Krieger sein Schwert schwingen.

# Warrior



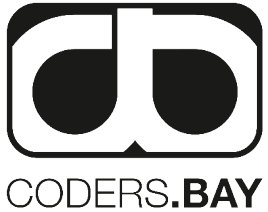

```
public class Warrior {

    private String name;
    private String gender;

    public Warrior(String name, String gender) {
        this.name = name;
        this.gender = gender;
    }

    public void swingSword() {
        System.out.println("pheeew \uD83D\uDDE1 ");
    }

}
```

# Wizard

CODERS.BAY

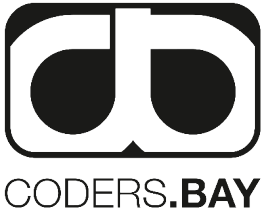
```
public class Wizard {

    private String name;
    private String gender;

    public Wizard(String name, String gender) {
        this.name = name;
        this.gender = gender;
    }

    public void makeMagic() {
        System.out.println("Magic! \uD83E\uDE84");
    }

}
```

# Elf




```java
public class Elf{

   private String name;
   private String gender;

   public Elf(String name, String gender) {
       this.name = name;
       this.gender = gender;
   }

   public void shootArrow() {
      System.out.println("--->    ");
   }

}
```

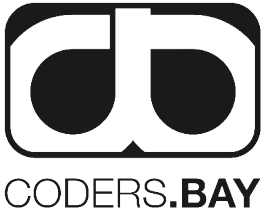
# Fighting




Wenn wir nun die Möglichkeit zum Kämpfen implementieren möchten, dann brauchen unsere Charaktere alle eine Variable, die die Gesundheit des Charakters ausdrückt.

Damit wir den Code nicht wieder kopieren müssen, extrahieren wir die gemeinsame Logik der drei Klassen in eine Klasse **RPGCharacter.**

# RPGCharacter

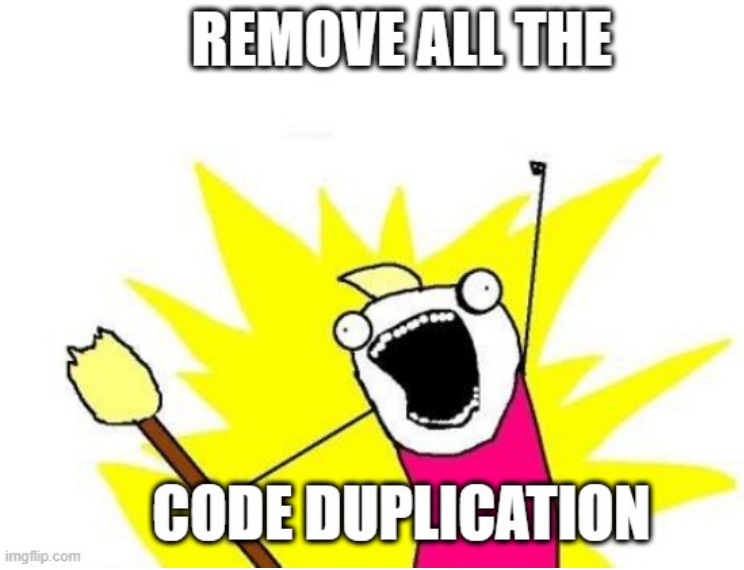



```
public abstract class RPGCharacter {

   private String name;
   private String gender;
   private Integer lifePoints;

   public RPGCharacter(String name,
                       String gender,
                       Integer lifePoints) {
       this.name = name;
       this.gender = gender;
       this.lifePoints = lifePoints;
   } // getter & setter
}
```

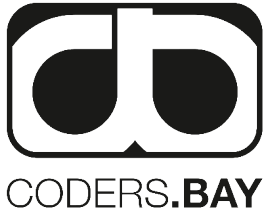
# RPGCharacter

CODERS.BAY

RPGCharacter kann nun vom Zauberer, Elf und Krieger erweitert werden um die Code Duplizierung zu vermindern.

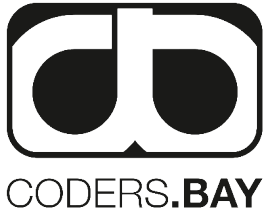


# Warrior



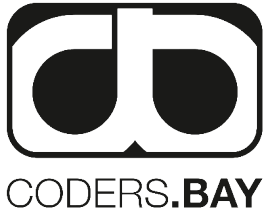

```
public class Warrior extends RPGCharacter {

    public Warrior(String name,
                   String gender,
                   Integer lifePoints) {
        super(name, gender, lifePoints);
    }

    public void swingSword() {
        System.out.println("pheeew \uD83D\uDDE1 ");
    }

}
```

# Wizard



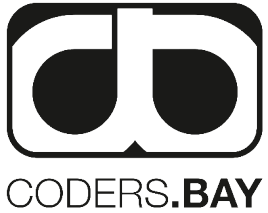

```
public class Wizard extends RPGCharacter {

    public Wizard(String name, String gender) {
        super(name, gender, 3);
    }

    public void makeMagic() {
        System.out.println("Magic! \uD83E\uDE84    ");
    }

}
```

# Elf




```
public class Elf extends RPGCharacter {

    public Elf(String name, String gender) {
        super(name, gender, Integer.MAX_VALUE);
    }

    public void shootArrow() {
        System.out.println("---> \uD83D\uDC9A");
    }

}
```

# Exkurs - Abstract Keyword

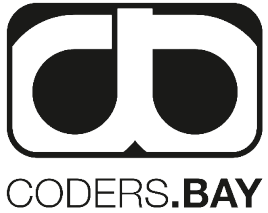



Das **abstract** Keyword bedeutet, dass eine Klasse abstrakt ist. Das heißt es können keine Instanzen von ihr erstellt werden!

Auch wenn RPGCharacter einen Konstruktor hat, kann dieser **nie direkt aufgerufen werden**, sondern nur von Klassen die RPGCharacter erweitern.

```
RPGCharacter character = new RPGCharacter("Pia", "female", 1337);
```

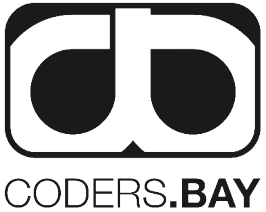
# Verwendung in einer Liste




In der Main Methode können wir uns jetzt eine Liste von RPG Charakteren anlegen und in diese Liste sowohl Krieger, Zauberer als auch Elfen hinzufügen.

Das funktioniert, weil durch die Vererbung gesagt ist, dass jeder Krieger, jeder Zauberer und jeder Elf auch als RPGCharacter gilt.

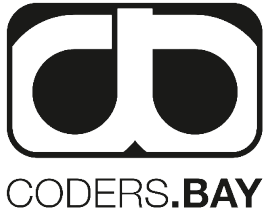
# main Methode




```
public static void main(String[] args) {
    List<RPGCharacter> allCharacters = new ArrayList<>();
    Warrior aragon = new Warrior("Aragon", "male", 10);
    Wizard gandalf = new Wizard("Gandalf the Grey", "male");
    Elf legolas = new Elf("Legolas", "male");
    allCharacters.add(aragon);
    allCharacters.add(gandalf);
    allCharacters.add(legolas);
    System.out.println(allCharacters);
}
```

```
[com.codersbay.heroes.Warrior@6acbcfc0, com.codersbay.heroes.Wizard@5f184fc6,
com.codersbay.heroes.elfs.Elf@3feba861]
```

# main Methode



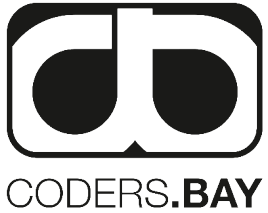

```
public static void mair
    List<RPGCharacter> a
    Warrior aragon = new
    Wizard gandalf = new
    Elf legolas = new El
    allCharacters.add(ar
    allCharacters.add(ga
    allCharacters.add(le
    System.out.println(a
}
```

Obwohl wir also die einzelnen Instanzen in einer Liste von RPGCharacter speichern, sehen wir durch die Ausgabe auf die Konsole dass die einzelnen Elemente sehr wohl noch ihren konkreten Typ haben!

Das heißt aragon wird zwar zu der Liste von Charakteren als RPGCharacter hinzugefügt, bleibt aber trotzdem noch ein Krieger genauso wie gandalf ein Zauberer bleibt.

```
[com.codersbay.heroes.Warrior@6acbcfc0, com.codersbay.heroes.Wizard@5f184fc6,
com.codersbay.heroes.elfs.Elf@3feba861]
```

# Zugriff auf Elemente in der Liste



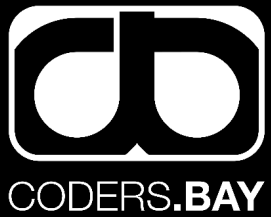

Wenn wir jetzt über die Elemente in der Liste allCharacters iterieren, dann haben wir das Problem, dass wir die jeweiligen Charaktere nicht als ihre konkrete Implementierung (Wizard, Warrior oder Elf) ansehen sondern als RPGCharacter.

Wenn wir jetzt zum Beispiel in einem Show-off unsere Charaktere ihre Fähigkeiten beweisen lassen wollen brauchen wir eine Möglichkeit auf den konkreten Typ zu schließen.

# Zugriff auf Elemente in der Liste

CODERS.BAY

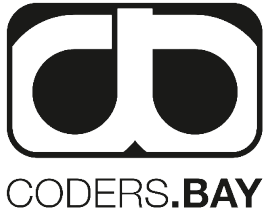
```
for (RPGCharacter character : allCharacters) {
    character.makeMagic();
    character.swingSword();
    character.shootArrow();
}
```




# Type Casts

Den Blickwinkel auf ein Objekt ändern

# Type Casts

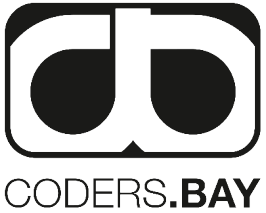



Was wir brauchen ist ein Type Cast. Mit einem Type Cast kann man seine Betrachtungsweise auf ein Objekt ändern. Casten kann man innerhalb einer Vererbungshierarchie.

Die Syntax ist einfach:

*(WantedType) variable*

# Type Casts



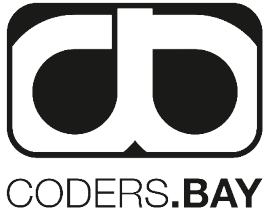

In der Vererbungshierachie **nach oben** zu Casten ist also einfach. Das heißt ein Elf, Krieger oder Zauberer kann jederzeit als RPGCharacter angeschaut werden.

In die Vererbungshierachie nach oben kann immer gecastet werden und das ganze passiert auch **implizit**:

Nicht notwendig, da implizit!

```
Elf legolas = new Elf("Legolas", "male");
RPGCharacter legolasCharacter = (RPGCharacter) legolas;
RPGCharacter legolasCharacter = legolas;
```

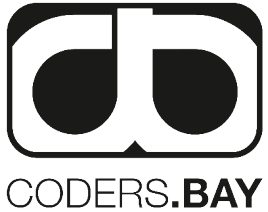
# Back to the topic




Zurück zu unserer for-Schleife vom Show-Off unserer Charaktere.
Zur Erinnerung wir haben eine List<RPGCharacter> allCharacters und möchten nun, dass jeder der Charaktere uns seine Fähigkeit beweist.
Mit einem Type Cast könnten wir soetwas machen:

```
for (RPGCharacter character : allCharacters) {
    ((Warrior) character).swingSword();
    ((Wizard) character).makeMagic();
    ((Elf) character).shootArrow();
}
```

# instanceof

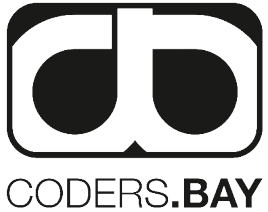



Das ergibt auch Sinn, denn wie sollte aragon denn auch ein Zauberer sein und Magie machen?

Was wir eigentlich wollen, ist nur jene Charaktere zu einem Krieger zu casten, die auch wirklich Krieger sind! Dafür gibt es den **instanceof** Operator. Dieser Operator gibt wahr zurück, wenn etwas eine Instanz einer bestimmten Klasse ist.

```
boolean isWarrior = character instanceof Warrior; // wahr für aragon
boolean isWizard = character instanceof Wizard; // wahr für gandalf
```

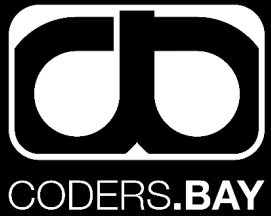
# instanceof




Wir können uns hiermit also helfen und unsere Charaktere, je nachdem was sie für ein Charakter sind, ihre Spezialfähigkeit beweisen lassen:

```
for (RPGCharacter character : allCharacters) {
    if (character instanceof Warrior) {
        ((Warrior) character).swingSword();
    } else if (character instanceof Wizard) {
        ((Wizard) character).makeMagic();
    } else if (character instanceof Elf) {
        ((Elf) character).shootArrow();
    }
}
```

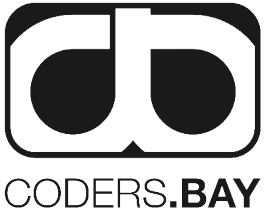
pheeew
Magic!
--->




# Abstrakte Methoden

Jeder Charakter muss die Fähigkeit showOff haben, aber nur ein konkreter Charakter weiß was er beim showOff zu tun hat.

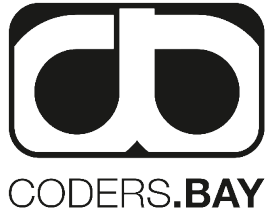
# ShowOff Improvement




Dieses Verhalten könnten wir noch verbessern, blöd ist es nämlich wenn wir dieses lange if-Konstrukt mit den instanceOf checks öfter brauchen.

Eine bessere Design-Entscheidung wäre es zum Beispiel, dem RPGCharacter eine abstrakte Methode showOff zu geben, mit dem jede Art von Character dann bestimmen kann, wie sie ihre Fähigkeiten beweisen würde.

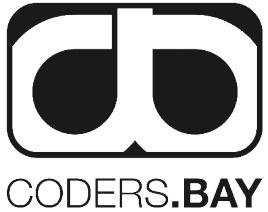
# Abstrakte Methoden




```
public abstract class RPGCharacter {

        // private fields
        // constructor
    public abstract void showOff();

}
```

Definieren wir eine Methode als abstrakt, darf sie keinen Method Body, also keine Logik in den geschwungenen Klammern haben!

# Abstract method must be implemented

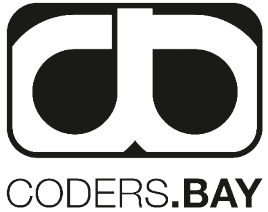



Jetzt haben wir unseren drei bestehenden Klassen Elf, Warrior und Wizard ein Problem:

*Class 'Warrior' must either be declared abstract or implement*
*abstract method 'showOff()' in 'RPGCharacter'*

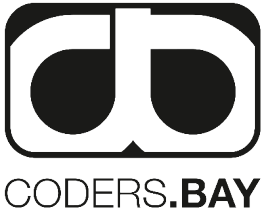
Das heißt jede Klasse, die RPGCharacter erweitert muss nun eine Methode showOff() definieren.

# Warrior



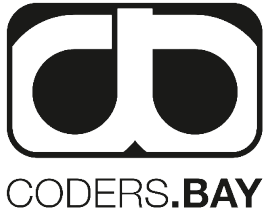

```
public class Warrior extends RPGCharacter {

    // constructor

    public void swingSword() {
        System.out.println("pheeew \uD83D\uDDE1 ");
    }

    @Override
    public void showOff() {
        swingSword();
    }
}
```

# Wizard



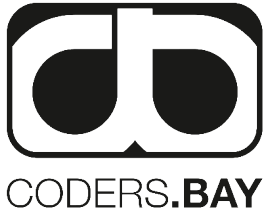

```
public class Wizard extends RPGCharacter {

    // constructor

    public void makeMagic() {
        System.out.println("Magic! \uD83E\uDE84    ");
    }

    @Override
    public void showOff() {
        makeMagic();
    }
}
```

# Elf




```
public class Elf extends RPGCharacter {

    // constructor

    public void shootArrow() {
        System.out.println("---> \uD83D\uDC9A");
    }

    @Override
    public void showOff() {
        shootArrow();
    }
}
```

# showOff




Jetzt können wir unsere showOff Logik der main Methode vereinfachen!
Viel weniger Zeilen Code und wir können auch jederzeit ganz einfach neue Rollenspiel Charaktere in unserem Spiel einführen.

```
for (RPGCharacter character : allCharacters) {
    character.showOff();
}
```

pheeew
Magic!
--->