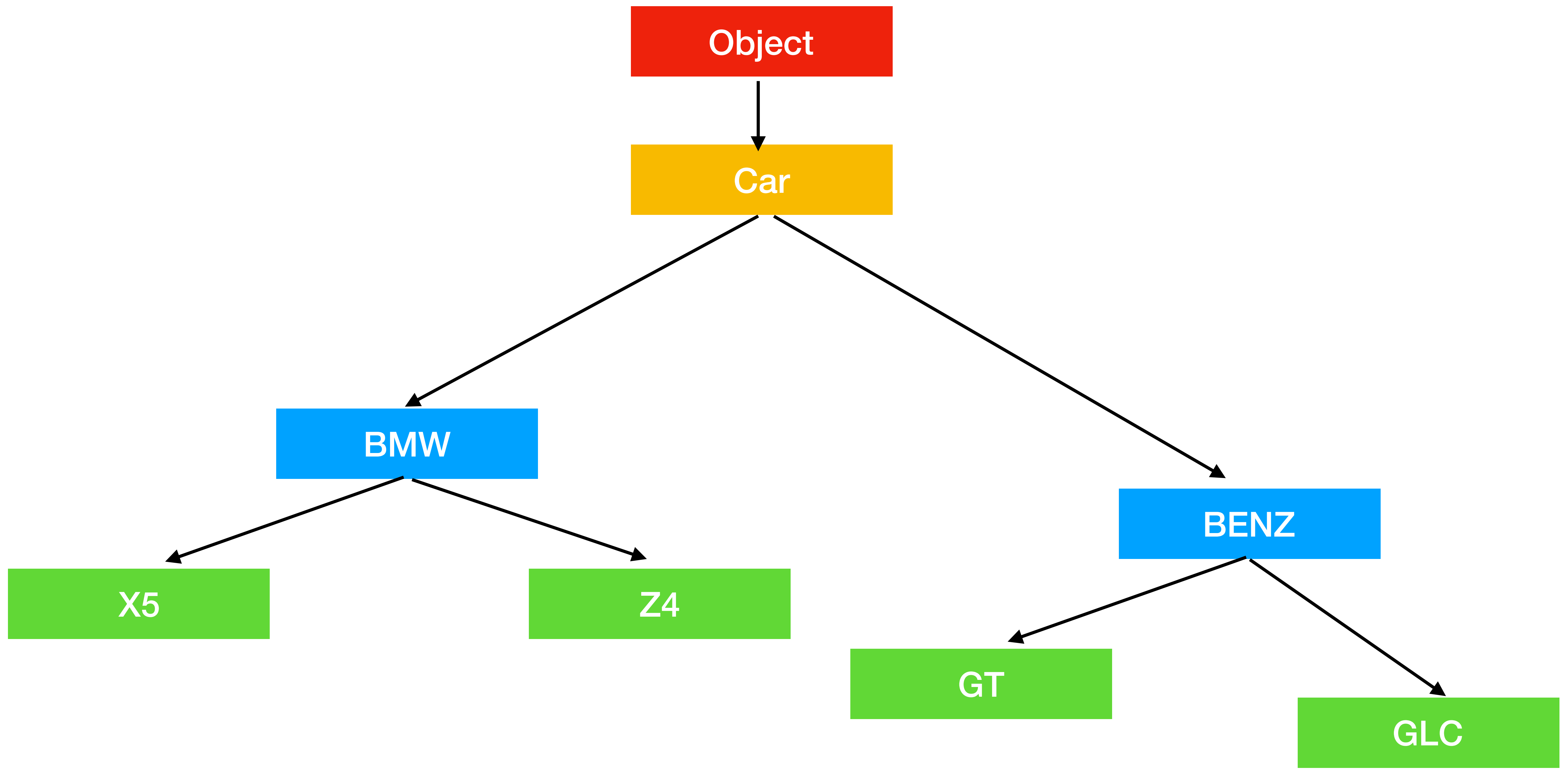
Java Parents to Child Class Abstract and Interface

2019 Lecture 3


Object
Car
BMW
BENZ
X5
Z4
GT
GLC

```java
package produce;

public class Factory {

    public static Car makeCar(String mode,  String branch) {
        if(mode.equals("BMW")) {
            if (branch.equals("X5")) {
                return new X5();
            } else if (branch.equals("Z4")) {
                return new Z4();
            } else {
                return null;
            }

        } else if (mode.equals("BENZ")) {
            if (branch.equals("GLC")) {
                return new GLC();
            } else if (branch.equals("GT")) {
                return new GT();
            } else {
                return null;
            }

        } else {
            return new Car("default", "default");
        }
    }
}
```

```java
package produce;

import Family.Parent;

public class Caller {
   public static void main(String[] args) {
      Car myNewCar = Factory.makeCar("BENZ", "GT");
      System.out.println(myNewCar);
   }
}
```

```
Benz GT AMG with V8 engine
```

Dynamic Binding
Late Binding

- Binding: Car c1 = new Car();
- Late Binding/Dynamic Binding: define which type to assign during run time

What if

- We don’t know what to define in the beginning
- We just have an abstraction of what is going on
- We just want to apply an enforcement

Problem 1
Meaningless function

```java
package game.nolimit;

public class HeroTemplate {
    public final String heroName;
    private int health;
    private int attack;

    public HeroTemplate(String name, int health, int attack) {
        this.heroName = name;
        this.health = health;
        this.attack = attack;
    }

    public void move() {
        System.out.print("Move up down right left");
    }

    public int normalAttack() {
        return attack * getCriticalHitRatio();
    }

    public void beingAttack(int hpCut) {
        health -= hpCut/getExtraArmarRatio();
    }

    public boolean isAlive() {
        return health > 0;
    }

    public void ultimateAttack() {

    }

    public int getCriticalHitRatio() {
        return 1;
    }

    public int getExtraArmarRatio() {
        return 1;
    }
}
```

Problem 2
No standard code

```java
package game.nolimit;

public class HeroTemplate2 {
    public final String heroName;
    private int health;
    private int attack;

    public HeroTemplate2(String name, int health, int attack) {
        this.heroName = name;
        this.health = health;
        this.attack = attack;
    }

    public void movemove() {
        System.out.print("Move up down right left");
    }

    public int normalAttack() {
        return attack;
    }

    public void beingAttack(int hpCut) {
        health -= hpCut/getExtraArmarRatio();
    }

    public boolean isAlive() {
        return health > 0;
    }

    public void ultimateAttack() {

    }

    public int getExtraArmarRatio() {
        return 1;
    }
}
```

Solve problem 1
Abstract class

- Class is declared as abstract
- Abstract class allows you not fully define function name
- BUT you can still use the function
- However you cannot initial an object from an abstract class
- Abstract class enforced the child class to implement the abstract method
- If child class does not know how to implement, declare abstract and parse to next lower level

```java
package game.nolimit;

abstract public class AbstractHeroTemplate {
    public final String heroName;
    private int health;
    private int attack;

    public AbstractHeroTemplate(String name, int health, int attack) {
        this.heroName = name;
        this.health = health;
        this.attack = attack;
    }

    public void move() {
        System.out.print("Move up down right left");
    }

    public int normalAttack() {
        return attack * getCriticalHitRatio();
    }

    public void beingAttack(int hpCut) {
        health -= hpCut;
    }

    public boolean isAlive() {
        return health > 0;
    }

    public void ultimateAttack() {

    }

    abstract public int getCriticalHitRatio();

    abstract public int getExtraArmarRatio();
}
```

```java
package game.nolimit;

public class Hero1 extends AbstractHeroTemplate{
   public Hero1(String name, int health) {
      super("Hero1", 100);
   }

   @Override
   public int getCriticalHitRatio() {
      return 2;
   }

   @Override
   public int getExtraArmarRatio() {
      return 2;
   }
}
```

Wrong initial

```
public static void main(String[] args) {
    AbstractHeroTemplate hero = new AbstractHeroTemplate();
}
```

Solve problem 2 Interface

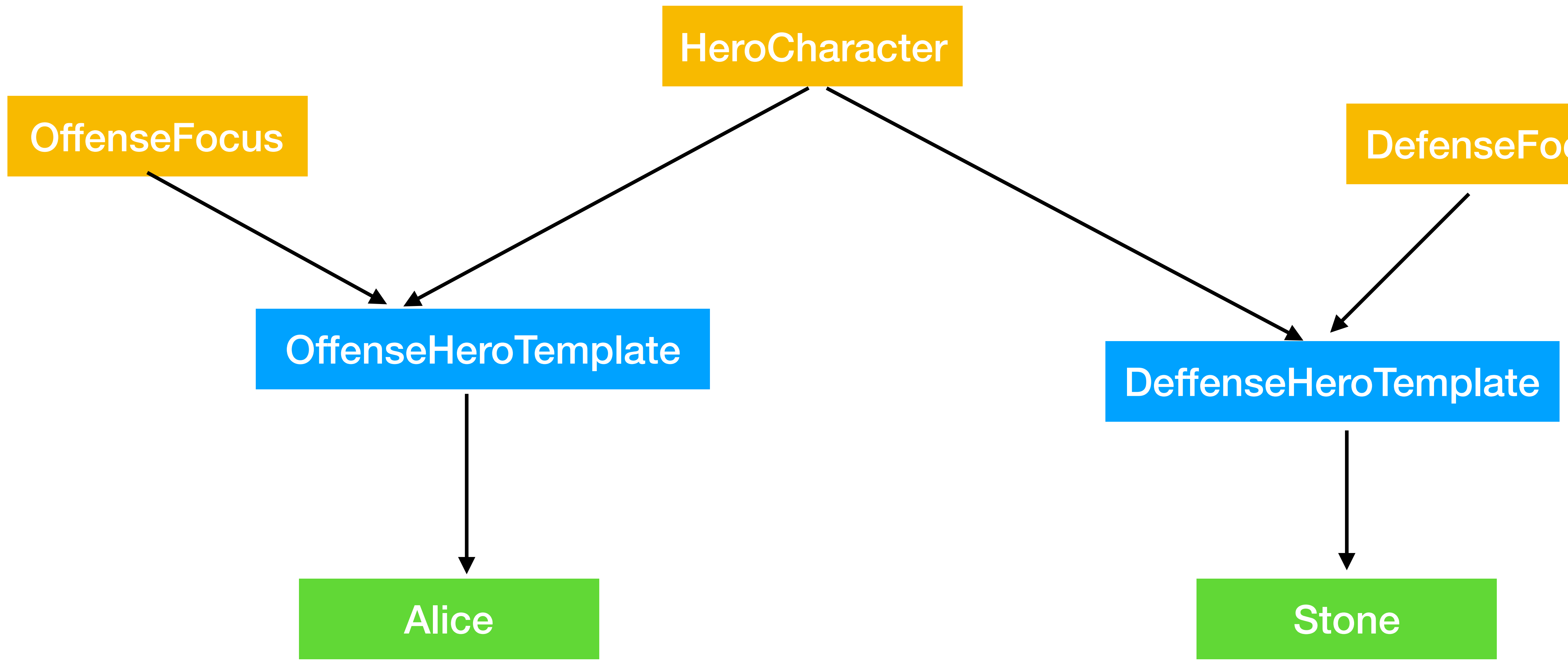
- Class is declared as interface
- Defines a standard
- All function in interface does not have a implementation body
- Interface define the basic function of class
- Interface enforce the class who implements it to implement all the function it defined
- Therefore all interface functions are public

```java
package game.nolimit;

public interface HeroCharacter {
   public void move();
   public int normalAttack();
   public void beingAttack(int hpCut);
   public boolean isAlive();
   public void ultimateAttack();
}
```

```java
package game.nolimit;

public class Hero2 implements HeroCharacter{
    public final String heroName;
    private int health;
    private int attack;

    public Hero2(String name, int health, int attack) {
        this.heroName = name;
        this.health = health;
        this.attack = attack;
    }

    @Override
    public void move() {
        System.out.print("Move up down right left");
    }

    @Override
    public int normalAttack() {
        return attack;
    }

    @Override
    public void beingAttack(int hpCut) {
        health -= hpCut;
    }

    @Override
    public boolean isAlive() {
        return health > 0;
    }

    @Override
    public void ultimateAttack() {
        System.out.print("ultimateAttack");
    }
}
```

Advanced structure

- A class can only extends one parent class
- Rule applies to abstract class too
- A class can implements unlimited interfaces
- interfaces provides a view, a list of characteristic, a different flavour of a class


HeroCharacter
OffenseFocus
DefenseFocus
OffenseHeroTemplate
DeffenseHeroTemplate
Alice
Stone

```java
package game;

public interface HeroCharacter {
    public void move();
    public int normalAttack();
    public void beingAttack(int hpCut);
    public boolean isAlive();
    public void ultimateAttack();
}
```

```java
package game;

public interface OffenseFocusHero {
    public int getCriticalHitRatio();
}
```

```java
package game;

public interface DefenseFocusHero {
    public int getExtraArmarRatio();
}
```

```java
package game;

abstract public class OffenseHeroTemplate implements HeroCharacter, OffenseFocusHero {
    public final String heroName;
    private int health;
    private int attack;

    public OffenseHeroTemplate(String name, int health, int attack) {
        this.heroName = name;
        this.health = health;
        this.attack = attack;
    }

    public void move() {
        System.out.print("Move up down right left");
    }

    public int normalAttack() {
        return attack * getCriticalHitRatio();
    }

    public void beingAttack(int hpCut) {
        health -= hpCut;
    }

    public boolean isAlive() {
        return health > 0;
    }

    abstract public void ultimateAttack();

    abstract public int getCriticalHitRatio();
}
```

```java
package game;

abstract public class DefenseHeroTemplate implements HeroCharacter, DefenseFocusHero {
    public final String heroName;
    private int health;
    private int attack;

    public DefenseHeroTemplate(String name, int health, int attack) {
        this.heroName = name;
        this.health = health;
        this.attack = attack;
    }

    public void move() {
        System.out.print("Move up down right left");
    }

    public int normalAttack() {
        return attack;
    }

    public void beingAttack(int hpCut) {
        health -= hpCut/getExtraArmarRatio();
    }

    public boolean isAlive() {
        return health > 0;
    }

    abstract public void ultimateAttack();

    abstract public int getExtraArmarRatio();

}
```

```java
package game;

public class AliceTheKiller extends OffenseHeroTemplate{

    public AliceTheKiller() {
        super("Alice", 50, 10);
    }

    @Override
    public void ultimateAttack() {
        System.out.print("Alice the killer ultimate-kill");
    }

    @Override
    public int getCriticalHitRatio() {
        return 2;
    }
}
```

```
package game;

public class StoneMan extends DefenseHeroTemplate{
    public StoneMan() {
        super("Stone man", 80, 5);
    }

    @Override
    public void ultimateAttack() {
        System.out.print("Stone man ultimate-kill");
    }

    @Override
    public int getExtraArmarRatio() {
        return 2;
    }
}
```

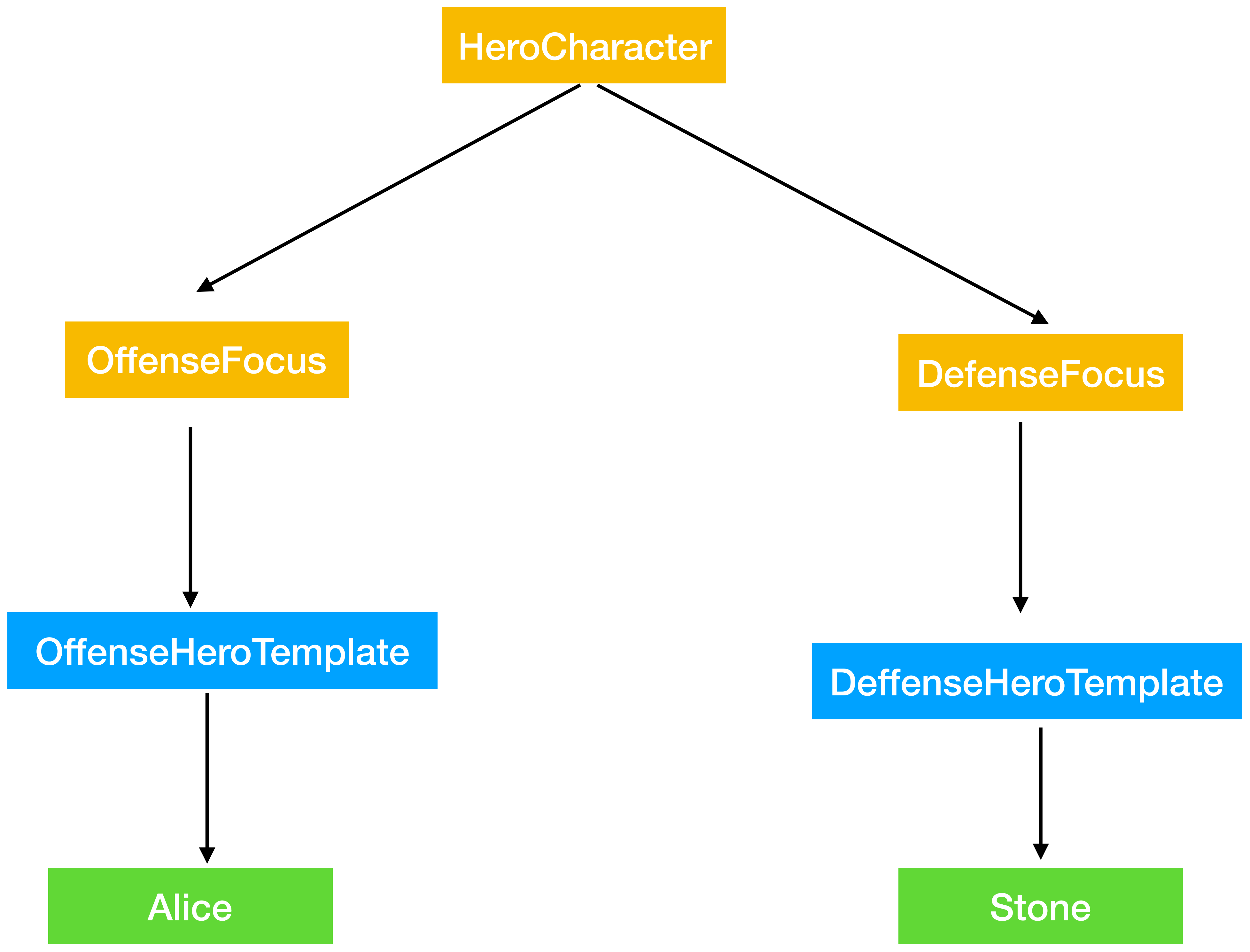
When game is called
The backend logic of
How hero is attacking
How hero get hit
Is well hidden

```java
package game;

import game.nolimit.AbstractHeroTemplate;

public class GameEngine {

    public static void main(String[] args) {

        HeroCharacter hero1 = chooseHero("Alice");

        hero1.move();
        hero1.normalAttack();
        hero1.ultimateAttack();
        hero1.beingAttack(3);

        HeroCharacter hero2 = chooseHero("Stone");

        hero2.move();
        hero2.normalAttack();
        hero2.ultimateAttack();
        hero2.beingAttack(6);
    }

    public static HeroCharacter chooseHero(String name) {
        if (name.equals("Alice")) {
            return new AliceTheKiller();
        } else if (name.equals("Stone")) {
            return new StoneMan();
        } else {
            return null;
        }
    }
}
```

```java
package game;

abstract public class HeroTemplate implements Move, Attack, UltiAttack, PurchaseItem{
}
```

Interface relationship

- Interface can extends other interface
- Since all interface function are public, the child interface will inherit all functions from parent interface
- It does not make sense for a interface to implement another interface
- Interface share the same rule of inheritance, can only extends from one interface


HeroCharacter
OffenseFocus
DefenseFocus
OffenseHeroTemplate
DeffenseHeroTemplate
Alice
Stone

```java
package game;

public interface HeroCharacter {
    public void move();
    public int normalAttack();
    public void beingAttack(int hpCut);
    public boolean isAlive();
    public void ultimateAttack();
}
```

```java
package game;

public interface OffenseFocusHero extends HeroCharacter {
    public int getCriticalHitRatio();
}
```

```java
package game;

public interface DefenseFocusHero extends HeroCharacter{
    public int getExtraArmarRatio();
}
```

```
package game;

abstract public class DefenseHeroTemplate implements DefenseFocusHero {
    public final String heroName;
    private int health;
    private int attack;

    public DefenseHeroTemplate(String name, int health, int attack) {
        this.heroName = name;
        this.health = health;
        this.attack = attack;
    }

    public void move() {
        System.out.print("Move up down right left");
    }

    public int normalAttack() {
        return attack;
    }

    public void beingAttack(int hpCut) {
        health -= hpCut/getExtraArmarRatio();
    }

    public boolean isAlive() {
        return health > 0;
    }

    abstract public void ultimateAttack();

    abstract public int getExtraArmarRatio();

}
```

Comparable interface

- Override equal function only allow the operation to compare if two object are same
- Implement Comparable interface allow the customized rule to compare two object

```java
public interface Comparable<T> {
    /**
     * Compares this object with the specified object for order.  Returns a
     * negative integer, zero, or a positive integer as this object is less
     * than, equal to, or greater than the specified object.
     *
     * <p>The implementor must ensure <tt>sgn(x.compareTo(y)) ==
     * -sgn(y.compareTo(x))</tt> for all <tt>x</tt> and <tt>y</tt>.  (This
     * implies that <tt>x.compareTo(y)</tt> must throw an exception iff
     * <tt>y.compareTo(x)</tt> throws an exception.)
     *
     * <p>The implementor must also ensure that the relation is transitive:
     * <tt>(x.compareTo(y)&gt;0 &amp;&amp; y.compareTo(z)&gt;0)</tt> implies
     * <tt>x.compareTo(z)&gt;0</tt>.
     *
     * <p>Finally, the implementor must ensure that <tt>x.compareTo(y)==0</tt>
     * implies that <tt>sgn(x.compareTo(z)) == sgn(y.compareTo(z))</tt>, for
     * all <tt>z</tt>.
     *
     * <p>It is strongly recommended, but <i>not</i> strictly required that
     * <tt>(x.compareTo(y)==0) == (x.equals(y))</tt>.  Generally speaking, any
     * class that implements the <tt>Comparable</tt> interface and violates
     * this condition should clearly indicate this fact.  The recommended
     * language is "Note: this class has a natural ordering that is
     * inconsistent with equals."
     *
     * <p>In the foregoing description, the notation
     * <tt>sgn(</tt><i>expression</i><tt>)</tt> designates the mathematical
     * <i>signum</i> function, which is defined to return one of <tt>-1</tt>,
     * <tt>0</tt>, or <tt>1</tt> according to whether the value of
     * <i>expression</i> is negative, zero or positive.
     *
     * @param   o the object to be compared.
     * @return  a negative integer, zero, or a positive integer as this object
     *          is less than, equal to, or greater than the specified object.
     *
     * @throws NullPointerException if the specified object is null
     * @throws ClassCastException if the specified object's type prevents it
     *         from being compared to this object.
     */
    public int compareTo(T o);
}
```

```java
public class Student implements Comparable{

    public int finalScore;
    public String name;
    public int grade;

    public Student(int finalScore, String name) {
        this.finalScore = finalScore;
    }

    public Student(int finalScore, String name, int grade) {
        this.finalScore = finalScore;
        this.name = name;
        this.grade = grade;
    }

    @Override
    public int compareTo(Object o) {
        if(grade == ((Student)o).grade)
            return 0;
        else if(grade > ((Student)o).grade)
            return 1;
        else
            return -1;
    }
}
```

```java
public class Student implements Comparable<Student>{

    public int finalScore;
    public String name;
    public int grade;

    public Student(int finalScore, String name) {
        this.finalScore = finalScore;
    }

    public Student(int finalScore, String name, int grade) {
        this.finalScore = finalScore;
        this.name = name;
        this.grade = grade;
    }

    @Override
    public int compareTo(Student other) {
        if(grade == other.grade)
            return 0;
        else if(grade > other.grade)
            return 1;
        else
            return -1;
    }
}
```

```java
public static void main(String[] args) {
    Student s1 = new Student(80, "Tom", 8);
    Student s2 = new Student(80, "Tim", 12);

    System.out.println(s1.compareTo(s2));
}
```

Print: -1