**Лабораторная работа №2**

**Задание 1.**

На основе выполненной работы в предыдущей практике и следующих требований, были сформулированы следующие структуры классов:

Абстрактный родительский класс GameObject:

abstract public class GameObject {  
 public String name;  
 public int id;   
}

Класс Creature расширяет GameObject и является родителем для всех *существ*:

public class Creatures extends GameObject {  
  
}

Класс Item расширяет GameObject и является базовым для любых предметов, *которыми может владеть игрок:*

public class Item extends GameObject{  
  
}

Класс Equipment расширяет базовый класс предметов и является родителем для любой *экипировки:*

public class Equipment extends Item {  
  
}

Класс EquipmentCell - класс-дженерик, который обрабатывает любые типы T расширяющие класс экипировки:

public class EquipmentCell<T> {  
  
 private T value = null;  
  
 public EquipmentCell(T value) {  
 this.value = value;  
 }  
  
 public T getValue() { return value; }  
}

Класс Inventory - класс-джереник, который обрабатывает типы T, расширяющие класс предметов:

public class Inventory<T extends Item> {  
  
 private List<T> list = new ArrayList<>();  
  
 public Inventory()  
 {  
  
 }  
 public Inventory(T value) {  
 list.add(value);  
 }  
  
 public List<T> getList() { return list; }  
  
 public void add(T value) {  
 list.add(value);  
 }  
  
 public void addAll(List<T> lst) {  
 list.addAll(lst);  
 }  
}

Интерфейс Lootable описывает поведение существ, *с которых после убийства падают предметы:*

public interface Lootable {

List<Item> dropLoot();  
}

Интерфейс Attacker описывает поведение сущности, *способной атаковать:*

public interface Attacker { void attack(Damageable target);  
}

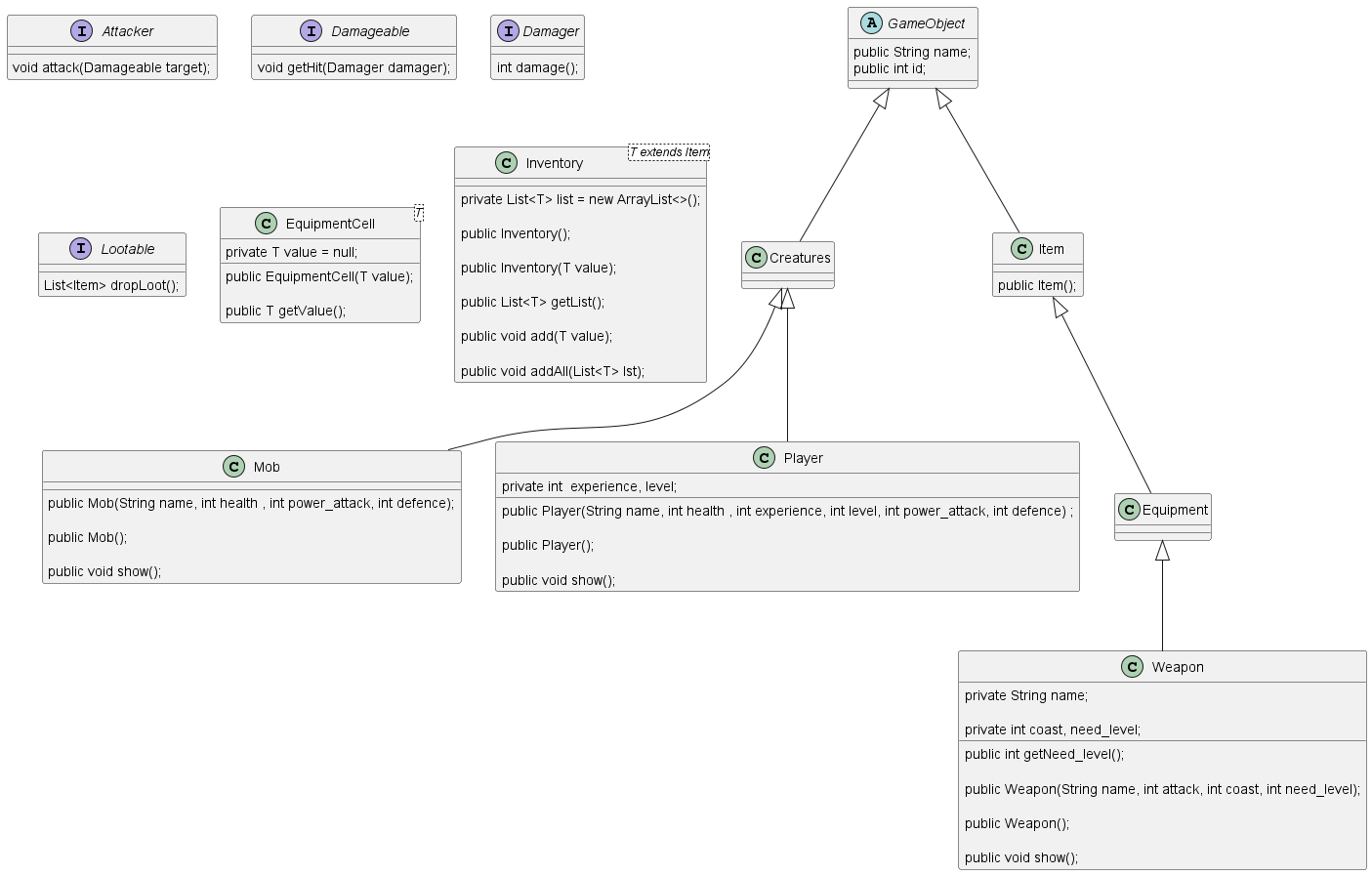
Интерфейс Damageable описывает сущности или объекты, которым *можно нанести урон:*

public interface Damageable { void getHit(Damager damager);  
}

Интерфейс Damager определяет поведение *предметов*, наносящих урон:

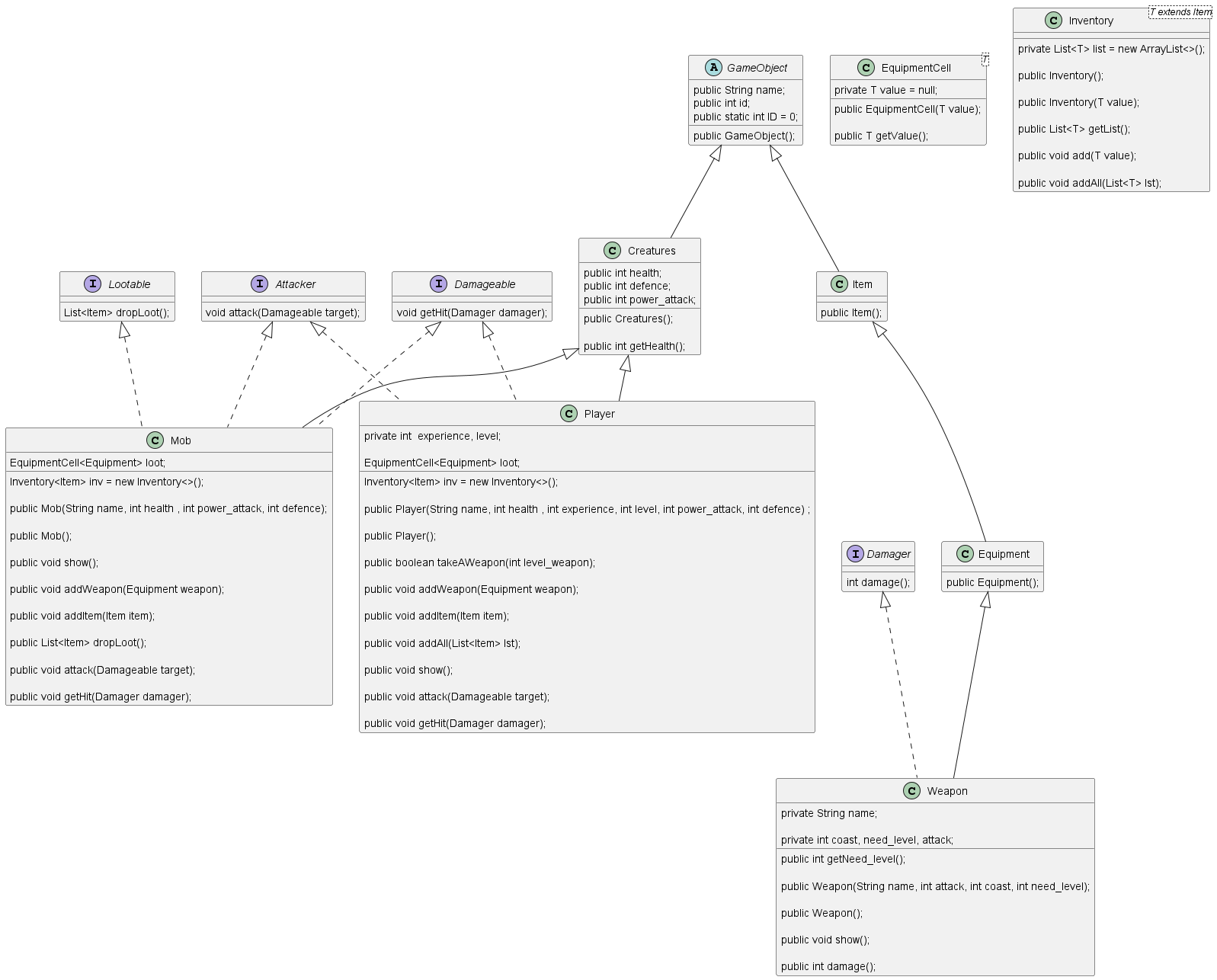
public interface Damager { int damage();  
}

Также были изменены названия классов: Character на Player, Enemy на Mob.



**Задание 2.**

Структуры классов были заполнены основными свойствами и поведением, соответствующие назначению классов, соблюдая логику наследования.



**Задание 3.**

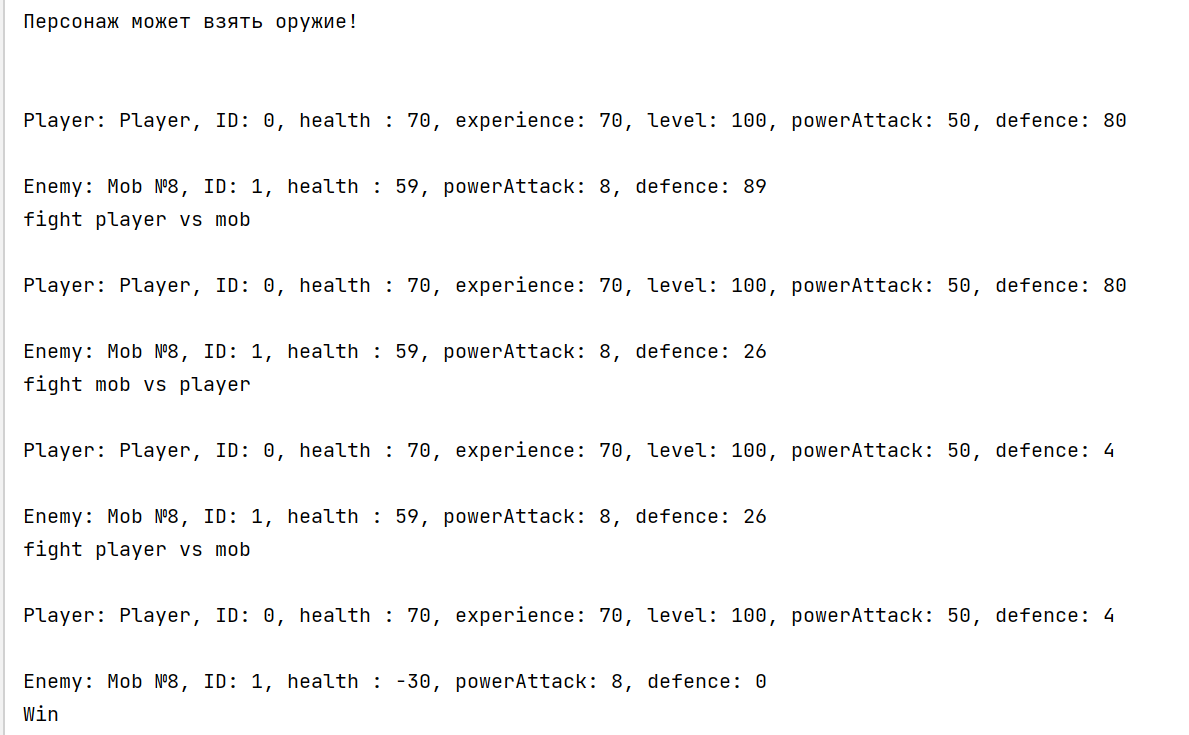
Были создан класс GameLogic для нанесение урона противнику:

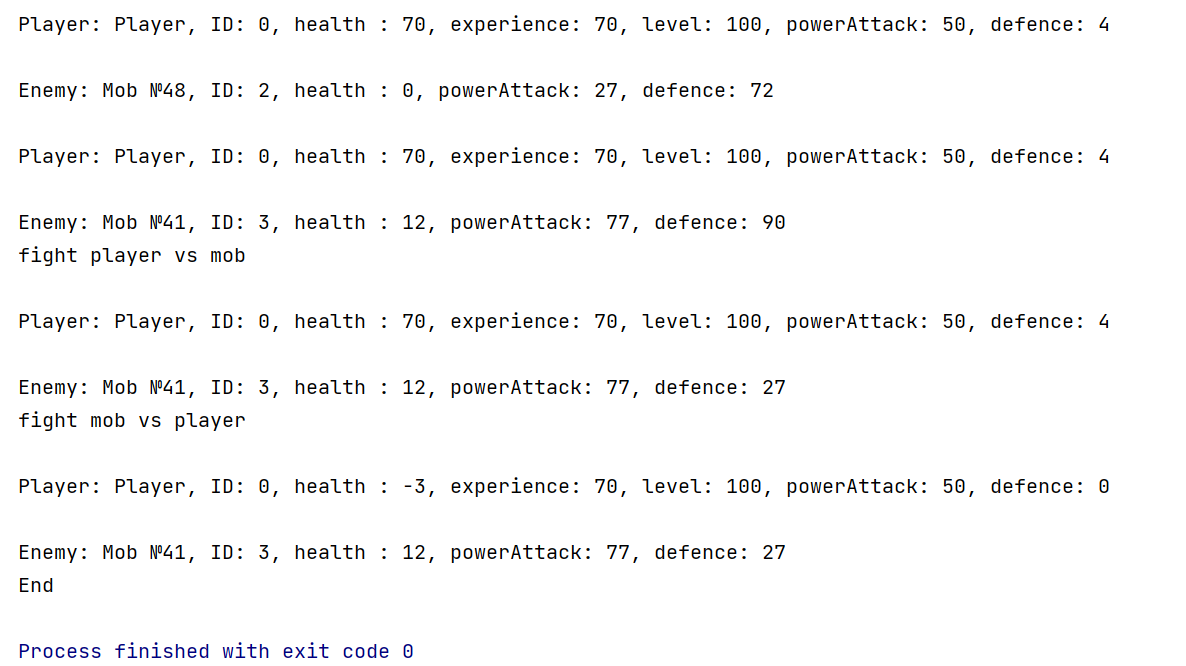
public class GameLogic {  
 static void fight(Attacker attacker, Damageable target) {  
 attacker.attack(target);  
 }  
}

И был создан класс GameLoop, в котором создаются игрок, мобы и оружие. И в методе run() происходит сражение игрока с мобами, пока игрок не победит всех или не умрет сам:

public class GameLoop {  
  
 static public Player *player* = new Player("Player", 70, 70, 100, 50, 80);  
 static public ArrayList<Mob> *masMob* = new ArrayList<>();  
 static public ArrayList<Weapon> *masWeapon* = new ArrayList<>();  
  
 public static void main(String[] args) {  
  
 *makeMob*(); *// создаем врагов  
 makeWeapons*(); *// создаем оружие  
  
 run*();  
  
 }  
  
 public static void run () {  
 Random rnd = new Random();  
 for(int i = 0; i < 10; i++) {  
 if(*player*.takeAWeapon(*masWeapon*.get(i).getNeed\_level())) {  
 *player*.addWeapon(*masWeapon*.get(i));  
 break;  
 }  
 }  
  
 for(Mob mob: *masMob*) {  
 *player*.show();  
 mob.show();  
 mob.addWeapon(*masWeapon*.get(rnd.nextInt((10) + 0)));  
 while (*player*.getHealth() > 0 && mob.getHealth() > 0) {  
 *fight*(*player*, mob);  
 System.*out*.println("fight player vs mob");  
 *player*.show();  
 mob.show();  
 if (mob.getHealth() > 0) {  
 *fight*(mob, *player*);  
 System.*out*.println("fight mob vs player");  
 *player*.show();  
 mob.show();  
 }  
 else {  
 System.*out*.println("Win");  
 *player*.addAll(mob.dropLoot());  
 break;  
 }  
 }  
  
 if(*player*.getHealth() <= 0) {  
 System.*out*.println("End");  
 break;  
 }  
 }  
 }  
  
 public static void makeMob() {  
 Random rnd = new Random();  
  
 for (int i = 0; i < 10; i++) {  
 String name = "Mob №" + rnd.nextInt((100) + 1);  
 int hp = rnd.nextInt((100) + 1);  
 int attack = rnd.nextInt((100) + 1);  
 int defence = rnd.nextInt((100) + 1);  
 *masMob*.add(new Mob(name, hp, attack, defence));  
 }  
 }  
  
 public static void makeWeapons() {  
 String[] names = {"Glock", "AK-47", "RX-22", "EP 22", "SPAS-12", "Stinger", "AT4", "RPG-7", "M9", "TMP"};  
 Random rnd = new Random();  
  
 for (int i = 0; i < 20; i++) {  
 String name = names[rnd.nextInt(10)];  
 int attack = rnd.nextInt((100) + 1);  
 int coast = rnd.nextInt((100) + 1);  
 int needLevel = rnd.nextInt((100) + 1);  
 *masWeapon*.add(new Weapon(name, attack, coast, needLevel));  
 }  
 }  
}

**Результат работы программы:**

****

****

**Приложение**:

**Пакет homework\_2.Logic**.

**Class GameLogic**

public class GameLogic {  
 static void fight(Attacker attacker, Damageable target) {  
 attacker.attack(target);  
 }  
}

**Class GameLoop**

public class GameLoop {  
  
 static public Player *player* = new Player("Player", 70, 70, 100, 50, 80);  
 static public ArrayList<Mob> *masMob* = new ArrayList<>();  
 static public ArrayList<Weapon> *masWeapon* = new ArrayList<>();  
  
 public static void main(String[] args) {  
  
 *makeMob*(); *// создаем врагов  
 makeWeapons*(); *// создаем оружие  
  
 run*();  
  
 }  
  
 public static void run () {  
 Random rnd = new Random();  
 for(int i = 0; i < 10; i++) {  
 if(*player*.takeAWeapon(*masWeapon*.get(i).getNeed\_level())) {  
 *player*.addWeapon(*masWeapon*.get(i));  
 break;  
 }  
 }  
  
 for(Mob mob: *masMob*) {  
 *player*.show();  
 mob.show();  
 mob.addWeapon(*masWeapon*.get(rnd.nextInt((10) + 0)));  
 while (*player*.getHealth() > 0 && mob.getHealth() > 0) {  
 *fight*(*player*, mob);  
 System.*out*.println("fight player vs mob");  
 *player*.show();  
 mob.show();  
 if (mob.getHealth() > 0) {  
 *fight*(mob, *player*);  
 System.*out*.println("fight mob vs player");  
 *player*.show();  
 mob.show();  
 }  
 else {  
 System.*out*.println("Win");  
 *player*.addAll(mob.dropLoot());  
 break;  
 }  
 }  
  
 if(*player*.getHealth() <= 0) {  
 System.*out*.println("End");  
 break;  
 }  
 }  
 }  
  
 public static void makeMob() {  
 Random rnd = new Random();  
  
 for (int i = 0; i < 10; i++) {  
 String name = "Mob №" + rnd.nextInt((100) + 1);  
 int hp = rnd.nextInt((100) + 1);  
 int attack = rnd.nextInt((100) + 1);  
 int defence = rnd.nextInt((100) + 1);  
 *masMob*.add(new Mob(name, hp, attack, defence));  
 }  
 }  
  
 public static void makeWeapons() {  
 String[] names = {"Glock", "AK-47", "RX-22", "EP 22", "SPAS-12", "Stinger", "AT4", "RPG-7", "M9", "TMP"};  
 Random rnd = new Random();  
  
 for (int i = 0; i < 20; i++) {  
 String name = names[rnd.nextInt(10)];  
 int attack = rnd.nextInt((100) + 1);  
 int coast = rnd.nextInt((100) + 1);  
 int needLevel = rnd.nextInt((100) + 1);  
 *masWeapon*.add(new Weapon(name, attack, coast, needLevel));  
 }  
 }  
}

**Пакет homework\_2.Structure**

**Class GameObject**

abstract public class GameObject {  
 public String name;  
 public int id;  
 public static int *ID* = 0;  
  
 public GameObject() {  
 id = *ID*;  
 *ID* += 1;  
 }  
}

**Class Creatures**

public class Creatures extends GameObject {  
 public int health;  
 public int defence;  
 public int powerAttack;  
  
 public Creatures() {  
 super();  
 }  
  
 public int getHealth() {  
 return health;  
 }  
  
}

**Class Item**

public class Item extends GameObject{  
  
 public Item() {  
 super();  
 }  
}

**Class Equipment**

public class Equipment extends Item {  
  
 public Equipment() {  
 super();  
 }  
  
}

**Class EquipmentCell<T>**

public class EquipmentCell<T> {  
  
 private T value = null;  
  
 public EquipmentCell(T value) {  
 this.value = value;  
 }  
  
 public T getValue() { return value; }  
}

**Class Inventory**

public class Inventory<T extends Item> {  
  
 private List<T> list = new ArrayList<>();  
  
 public Inventory()  
 {  
  
 }  
 public Inventory(T value) {  
 list.add(value);  
 }  
  
 public List<T> getList() { return list; }  
  
 public void add(T value) {  
 list.add(value);  
 }  
  
 public void addAll(List<T> lst) {  
 list.addAll(lst);  
 }  
}

**interface Attacker**

public interface Attacker { *//описывает поведение сущности, способной атаковать* void attack(Damageable target);  
}

**interface Damager**

public interface Damager { *//определяет поведение предметов, наносящих урон* int damage();  
}

**interface Damageable**

public interface Damageable { *//описывает сущности или объекты, которым можно нанести урон* void getHit(Damager damager);  
}

**interface Lootable**

public interface Lootable { *//поведение существ, с которых после убийства падают предметы* List<Item> dropLoot();  
}

**Пакет homework\_2**

**Class Player**

public class Player extends Creatures implements Attacker, Damageable {  
 private int experience, level;  
 EquipmentCell<Equipment> loot = null;  
 Inventory<Item> inv = new Inventory<>();  
  
 public Player(String name, int health , int experience, int level, int powerAttack, int defence) {  
 super();  
 this.name = name;  
 this.health = health ;  
 this.experience = experience;  
 this.level = level;  
 this.powerAttack = powerAttack;  
 this.defence = defence;  
 }  
  
 public Player() {  
 name = "Hero";  
 health = id = experience = level = powerAttack = defence = 0;  
 }  
  
 public boolean takeAWeapon(int levelWeapon) {  
  
 if (level >= levelWeapon) {  
 System.*out*.println("Персонаж может взять оружие!\n");  
 return true;  
 } else {  
 System.*out*.println("Персонаж не может поднять оружие!\n");  
 return false;  
 }  
 }  
  
 public void addWeapon(Equipment weapon) {  
 loot = new EquipmentCell<>(weapon);  
 addItem(weapon);  
 }  
  
 public void addItem(Item item) {  
 inv.add(item);  
 }  
  
 public void addAll(List<Item> lst) {  
 inv.addAll(lst);  
 }  
  
 public void show() {  
 System.*out*.println(  
 "\nPlayer: " + name + ", ID: " + id + ", health : " + health + ", experience: " + experience  
 + ", level: " + level + ", powerAttack: " + powerAttack + ", defence: " + defence  
 );  
 }  
  
 @Override  
 void attack(Damageable target) {  
 if(Damager.class.isAssignableFrom(loot.getValue().getClass())

) {  
 Damager damager = (Damager) loot.getValue();  
 target.getHit(damager);  
 }  
 else {  
 System.*out*.println("У атакующего нет оружия");  
 }  
 }

@Override  
 public void getHit(Damager damager) {  
 if(defence - damager.damage() > 0)  
 defence -= damager.damage();  
 else {  
 health -= damager.damage() + defence;  
 defence = 0;  
 }  
 }  
  
}

**Class Mob**

public class Mob extends Creatures implements Lootable, Attacker, Damageable {  
 EquipmentCell<Equipment> loot;  
 Inventory<Item> inv = new Inventory<>();  
  
 public Mob(String name, int health , int powerAttack, int defence) {  
 super();  
 this.name = name;  
 this.health = health ;  
 this.id = id;  
 this.powerAttack = powerAttack;  
 this.defence = defence;  
 }  
  
 public Mob() {  
 name = "Enemy";  
 health = id = powerAttack = defence = 0;  
 }  
  
 public void show() {  
 System.*out*.println(  
 "\nEnemy: " + name + ", ID: " + id + ", health : " + health + ", powerAttack: " +  
 powerAttack + ", defence: " + defence  
 );  
 }  
  
 public void addWeapon(Equipment weapon) {  
 loot = new EquipmentCell<>(weapon);  
 addItem(weapon);  
 }  
  
 public void addItem(Item item) {  
 inv.add(item);  
 }  
  
 @Override  
 public List<Item> dropLoot() {  
 return inv.getList();  
 }  
  
 @Override  
 public void attack(Damageable target) {  
 if(Damager.class.isAssignableFrom(loot.getValue().getClass())) {  
 Damager damager = (Damager) loot.getValue();  
 target.getHit(damager);  
 }  
 else {  
 System.*out*.println("У атакующего нет оружия");  
 }  
 }

@Override  
 public void getHit(Damager damager) {  
 if(defence - damager.damage() > 0)  
 defence -= damager.damage();  
 else {  
 health -= damager.damage() + defence;  
 defence = 0;  
 }  
 }  
  
}

**Class Weapon**

public class Weapon extends Equipment implements Damager {  
  
 private String name;  
 private int coast, needLevel, attack;  
  
 public int getNeed\_level() {  
 return needLevel;  
 }  
  
 public Weapon(String name, int attack, int coast, int needLevel) {  
 super();  
 this.name = name;  
 this.attack = attack;  
 this.coast = coast;  
 this.needLevel = needLevel;  
 }  
  
 public Weapon() {  
 name = "Weapon";  
 attack = coast = needLevel = 0;  
 }  
  
  
 public void show() {  
 System.*out*.println(  
 "\nWeapon: " + name + ", ID: " + id + ", attack: " + attack +  
 ", coast: " + coast + ", needLevel: " + needLevel  
 );  
 }

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