Факультет Систем Управления и Робототехники

Лабораторная работа №2

по Программированию

Вариант 17499

Выполнил:

Рязанов Даниил Дмитриевич

Группа R3140

Преподаватель:

Заболотняя Ольга Михайловна

Содержание

[Текст задания 3](#_Toc212003050)

[Диаграмма классов реализованной объектной модели 5](#_Toc212003051)

[Исходный код программы 5](#_Toc212003052)

[Упаковка и запуск программы 8](#_Toc212003053)

[Результат работы программы 8](#_Toc212003054)

[Выводы по работе 9](#_Toc212003055)

[Приложение 10](#_Toc212003056)

# Текст задания

На основе базового класса Pokemon написать свои классы для заданных видов покемонов. Каждый вид покемона должен иметь один или два типа и стандартные базовые характеристики:

* очки здоровья (HP)
* атака (attack)
* защита (defense)
* специальная атака (special attack)
* специальная защита (special defense)
* скорость (speed)

Классы покемонов должны наследоваться в соответствии с цепочкой эволюции покемонов. На основе базовых классов PhysicalMove, SpecialMove и StatusMove реализовать свои классы для заданных видов атак. Все разработанные классы, не имеющие наследников, должны быть реализованы таким образом, чтобы от них нельзя было наследоваться.

Атака должна иметь стандартные тип, силу (power) и точность (accuracy). Должны быть реализованы стандартные эффекты атаки. Назначить каждому виду покемонов атаки в соответствии с вариантом. Уровень покемона выбирается минимально необходимым для всех реализованных атак.

Используя класс симуляции боя Battle, создать 2 команды покемонов (каждый покемон должен иметь имя) и запустить бой.

Базовые классы и симулятор сражения находятся в jar-архиве (обновлен 9.10.2018, исправлен баг с добавлением атак и кодировкой). Документация в формате javadoc - здесь.

Информацию о покемонах, цепочках эволюции и атаках можно найти на сайтах http://poke-universe.ru, http://pokemondb.net, http://veekun.com/dex/pokemon

**Комментарии**

Цель работы: на простом примере разобраться с основными концепциями ООП и научиться использовать их в программах.

**Что надо сделать (краткое описание)**

Ознакомиться с документацией, обращая особое внимание на классы Pokemon и Move. При дальнейшем выполнении лабораторной работы читать документацию еще несколько раз.

Скачать файл Pokemon.jar. Его необходимо будет использовать как для компиляции, так и для запуска программы. Распаковывать его не надо! Нужно научиться подключать внешние jar-файлы к своей программе.

Написать минимально работающую программу и посмотреть как она работает.

Battle b = new Battle();

Pokemon p1 = new Pokemon("Чужой", 1);

Pokemon p2 = new Pokemon("Хищник", 1);

b.addAlly(p1);

b.addFoe(p2);

b.go();

Создать один из классов покемонов для своего варианта. Класс должен наследоваться от базового класса Pokemon. В конструкторе нужно будет задать типы покемона и его базовые характеристики. После этого попробуйте добавить покемона в сражение.

Создать один из классов атак для своего варианта (лучше всего начать с физической или специальной атаки). Класс должен наследоваться от класса PhysicalMove или SpecialMove. В конструкторе нужно будет задать тип атаки, ее силу и точность. После этого добавить атаку покемону и проверить ее действие в сражении. Не забудьте переопределить метод describe, чтобы выводилось нужное сообщение.

Если действие атаки отличается от стандартного, например, покемон не промахивается, либо атакующий покемон также получает повреждение, то в классе атаки нужно дополнительно переопределить соответствующие методы (см. документацию). При реализации атак, которые меняют статус покемона (наследники StatusMove), скорее всего придется разобраться с классом Effect. Он позволяет на один или несколько ходов изменить состояние покемона или модификатор его базовых характеристик.

Доделать все необходимые атаки и всех покемонов, распределить покемонов по командам, запустить сражение.

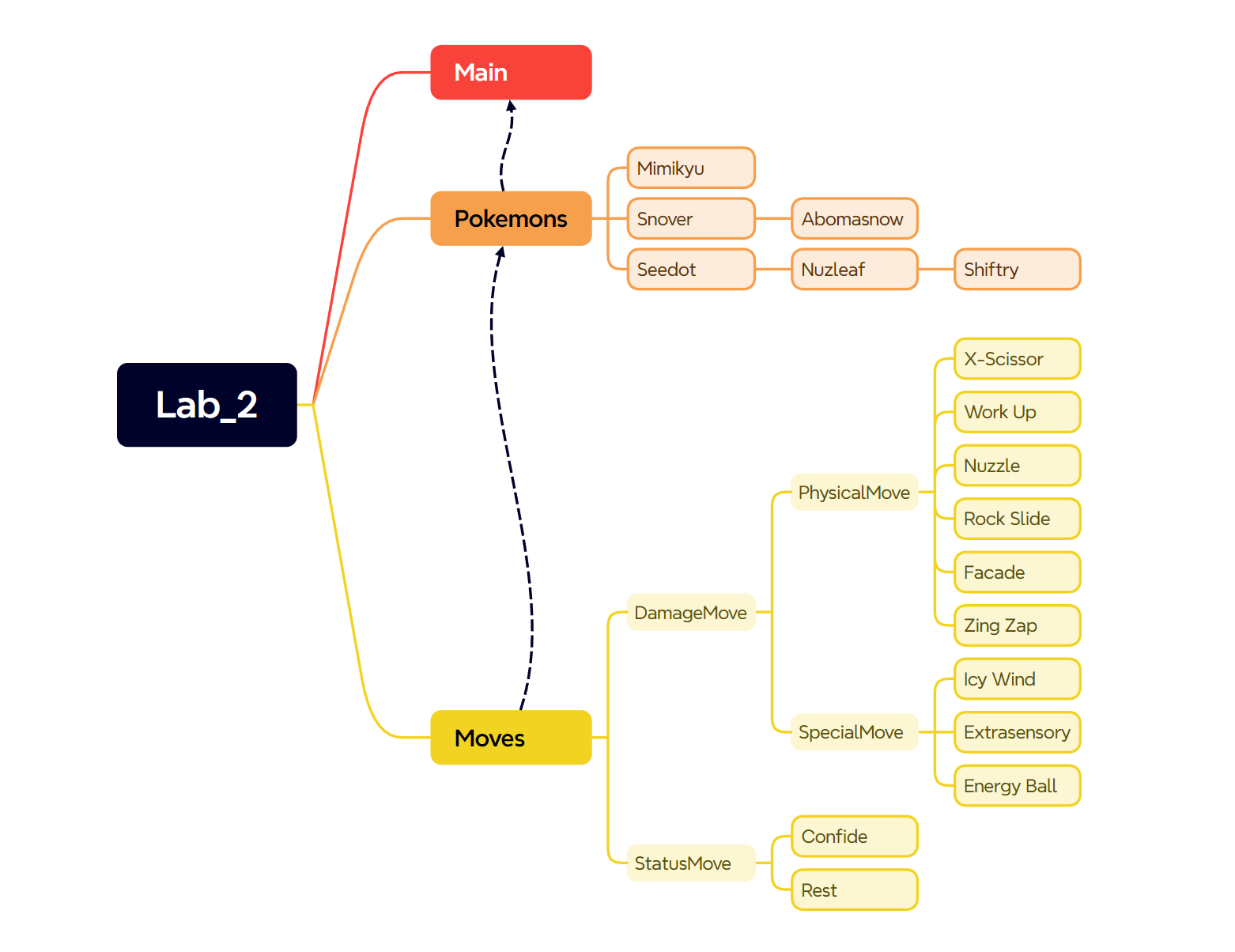


# Диаграмма классов реализованной объектной модели

Архитектура классов разделена на два пакета: Pokemons и Moves для покемонов и атак соответственно.

Все покемоны наследуются от суперкласса Pokemon. Покемоны организованы в цепочки наследования, в которых каждый покемон наследует характеристики предка и дополняет их собственными.

Атаки наследуются от соответствующих их типам подклассов.



# Исходный код программы

В функции main создается объект класса, реализующего битву покемонов. Затем формируются две команды, состоящие из разработанных покемонов. После этого запускается битва.

Исходный код файла Main.java (код из остальных файлов представлен в приложении):

import ru.ifmo.se.pokemon.\*;

import Pokemons.\*;

public class Main {

    public static void main(String[] args) {

        Battle battle = new Battle();

        battle.addAlly(new Snover("Frostbite", 1));

        battle.addAlly(new Abomasnow("Blizzard", 1));

        battle.addAlly(new Seedot("Acorn", 1));

        battle.addFoe(new Mimikyu("Disguise", 1));

        battle.addFoe(new Nuzleaf("Shadowleaf", 1));

        battle.addFoe(new Shiftry("Stormwind", 1));

        battle.go();

    }

}

Для реализации каждого покемона создавался отдельный класс, наследуемый либо от суперкласса Pokemon, либо от предыдущего производного класса в цепочке наследования. Затем реализовывался конструктор, в котором вызывался конструктор базового класса, после чего устанавливались характеристики покемона, тип и добавлялись его атаки. Если покемон имел предков, то вместо set-функций использовались add-функции, которые расширяли унаследованные характеристики.

package Pokemons;

import ru.ifmo.se.pokemon.Pokemon;

import ru.ifmo.se.pokemon.Type;

import Moves.\*;

public class Seedot extends Pokemon {

    public Seedot (String name, int level) {

        super(name, level);

        /\*

         \* HP               40

         \* Attack           40

         \* Defense          50

         \* Special Attack   30

         \* Spacial Defense  30

         \* Speed            30

         \*/

        this.setStats(40d, 40d, 50d, 30d, 30d, 30d);

        this.setType(Type.GRASS);

        this.setMove(new EnergyBall(), new Confide());

    }

}

Для реализации атак, в зависимости от их свойств, применялась перегрузка методов applySelfEffects и applyOppEffects. В них создавался объект класса Effect, с помощью которого изменялись необходимые характеристики покемонов. Все атаки реализованы через механику класса Effect для обеспечения единообразия в изменении параметров покемонов.

package Moves;

import ru.ifmo.se.pokemon.\*;

public final class Nuzzle extends PhysicalMove {

    public Nuzzle() {

        /\*

         \* power 20

         \* accuracy 100

         \* priority 0

         \* hits 1

         \*/

        super(Type.ELECTRIC, 20, 100, 0, 1);

    }

    /\*

     \* Nuzzle deals damage and paralyzes the target. Paralyzed Pokémon have a 25%

     \* chance of not being able to attack, and their Speed is decreased by 50% (75%

     \* in Generations 1-6).

     \*

     \* Electric type Pokémon, those with the ability Limber or those behind a

     \* Substitute cannot be paralyzed.

     \*/

    @Override

    protected void applyOppEffects(Pokemon p) {

        p.addEffect(new Effect().condition(Status.PARALYZE));

    }

    @Override

    protected String describe() {

        return "uses Nuzzle";

    }

}

Все классы, не предусматривающие наследования по условию, помечены final для запрета наследования.

# Упаковка и запуск программы

Исходный код программы расположен в папке src.

Для упаковки и запуска программы написан shell-скрипт autobuild.sh, который выполняет компилирование программы, упаковку байт-кода в jar-архив и его запуск. Исходный код файла autobuild.sh:

mkdir -p build

cd src

javac -d "../build/" -cp Pokemon.jar Main.java Pokemons/\*.java Moves/\*.java

cp Pokemon.jar ../build/

cp main.mf ../build/

cd ../build

jar -cfm Lab\_2.jar main.mf \*.class Pokemons/\*.class Moves/\*.class

java -Xmx512m -XX:MaxMetaspaceSize=128m -jar Lab\_2.jar

# Результат работы программы

Snover Frostbite from the team Black enters the battle!

Mimikyu Disguise from the team Yellow enters the battle!

Mimikyu Disguise uses Nuzzle.

Snover Frostbite loses 4 hit points.

Snover Frostbite is paralyzed

Snover Frostbite uses Energy Ball.

Mimikyu Disguise loses 4 hit points.

Mimikyu Disguise uses Facade.

Snover Frostbite loses 6 hit points.

Snover Frostbite increases attack.

Snover Frostbite misses

Mimikyu Disguise misses

Snover Frostbite misses

Mimikyu Disguise misses

Snover Frostbite uses Icy Wind.

Mimikyu Disguise loses 5 hit points.

Mimikyu Disguise uses Nuzzle.

Snover Frostbite loses 3 hit points.

Snover Frostbite faints.

Abomasnow Blizzard from the team Black enters the battle!

Mimikyu Disguise uses Zing Zap.

Abomasnow Blizzard loses 4 hit points.

Abomasnow Blizzard uses Rock Slide.

Mimikyu Disguise loses 5 hit points.

Mimikyu Disguise faints.

Nuzleaf Shadowleaf from the team Yellow enters the battle!

Nuzleaf Shadowleaf uses Energy Ball.

Abomasnow Blizzard loses 5 hit points.

Abomasnow Blizzard uses Rock Slide.

Abomasnow Blizzard loses 9 hit points.

Both pokemons faint.

Seedot Acorn from the team Black enters the battle!

Nuzleaf Shadowleaf uses Extrasensory.

Seedot Acorn loses 4 hit points.

Seedot Acorn uses Energy Ball.

Nuzleaf Shadowleaf loses 4 hit points.

Nuzleaf Shadowleaf misses

Nuzleaf Shadowleaf uses Extrasensory.

Seedot Acorn loses 4 hit points.

Nuzleaf Shadowleaf uses Extrasensory.

Seedot Acorn loses 4 hit points.

Seedot Acorn faints.

Team Black loses its last Pokemon.

The team Yellow wins the battle!

# Выводы по работе

В ходе выполнения данной лабораторной работы были изучены и применены на практике ключевые концепции объектно-ориентированного программирования на языке Java. Основные навыки, приобретенные в процессе работы:

1. Создание и работа с классами в Java — научился разрабатывать собственные классы, определять их структуру, поля и методы, а также создавать конструкторы для инициализации объектов.
2. Использование наследования — освоил принципы наследования классов, создание иерархий наследования, переопределение методов и использование ключевого слова super для обращения к базовому классу.
3. Работа с документацией — получил опыт работы с внешней документацией библиотек, изучения API и применения готовых классов и методов в соответствии с их спецификацией.
4. Подключение JAR-архивов — научился подключать скомпилированные JAR-архивы к проекту, настраивать classpath и использовать внешние библиотеки в собственных проектах.
5. **Разработка программной архитектуры** - получил практический опыт проектирования архитектуры приложения, разделения функциональности на логические модули и пакеты, а также организации кода в соответствии с принципами объектно-ориентированного программирования.
6. **Работа с полиморфизмом** - освоил применение полиморфизма через переопределение методов в производных классах и использование общих интерфейсов для различных типов объектов.
7. **Организация кода в пакеты** - научился структурировать код в пакеты для логического разделения функциональности и улучшения читаемости проекта.
8. **Создание исполняемых JAR-файлов** - получил опыт сборки проектов в исполняемые JAR-архивы с помощью утилиты jar и настройки манифест-файлов.

В результате была создана полнофункциональная система боев покемонов, демонстрирующая применение всех изученных концепций и принципов объектно-ориентированного программирования. Работа позволила закрепить теоретические знания на практик

# Приложение

|  |
| --- |
| package Pokemons;  import Moves.\*;  public final class Abomasnow extends Snover {      public Abomasnow(String name, int level) {          super(name, level);          /\*           \* HP               90           \* Attack           92           \* Defense          75           \* Special Attack   92           \* Spacial Defense  85           \* Speed            60           \*/          this.setStats(90d, 92d, 75d, 92d, 85d, 60d);            // Add Rock Slide to legacy attacks (Rest, Icy Wind, Energy Ball)          this.addMove(new RockSlide());      }    } |
| package Pokemons;  import ru.ifmo.se.pokemon.Pokemon;  import ru.ifmo.se.pokemon.Type;  import Moves.\*;  public final class Mimikyu extends Pokemon {      public Mimikyu(String name, int level) {       super(name, level);          /\*           \* HP               55           \* Attack           90           \* Defense          80           \* Special Attack   50           \* Spacial Defense  105           \* Speed            96           \*/          this.setStats(55d, 90d, 80d, 50d, 105d, 96d);          this.setType(Type.FAIRY);          this.setMove(new ZingZap(), new WorkUp(), new Facade(), new Nuzzle());      }    } |
| package Pokemons;  import ru.ifmo.se.pokemon.Pokemon;  import ru.ifmo.se.pokemon.Type;  import Moves.\*;  public class Nuzleaf extends Seedot {      public Nuzleaf(String name, int level) {       super(name, level);          /\*           \* HP               70           \* Attack           70           \* Defense          40           \* Special Attack   60           \* Spacial Defense  40           \* Speed            60           \*/          this.setStats(70d, 70d, 40d, 60d, 40d, 60d);          this.addType(Type.DARK);            // Add Extrasensory to legacy attacks (Energy Ball, Confide)          this.addMove(new Extrasensory());      }  } |
| package Pokemons;  import ru.ifmo.se.pokemon.Pokemon;  import ru.ifmo.se.pokemon.Type;  import Moves.\*;  public class Seedot extends Pokemon {      public Seedot (String name, int level) {          super(name, level);          /\*           \* HP               40           \* Attack           40           \* Defense          50           \* Special Attack   30           \* Spacial Defense  30           \* Speed            30           \*/          this.setStats(40d, 40d, 50d, 30d, 30d, 30d);          this.setType(Type.GRASS);          this.setMove(new EnergyBall(), new Confide());      }  } |
| package Pokemons;  import Moves.\*;  public final class Shiftry extends Nuzleaf {      public Shiftry(String name, int level) {       super(name, level);          /\*           \* HP               90           \* Attack           100           \* Defense          60           \* Special Attack   90           \* Spacial Defense  60           \* Speed            80           \*/          this.setStats(90d, 100d, 60d, 90d, 60d, 80d);          // Add X-Scissor to legacy attacks (Energy Ball, Confide, Extrasensory)          this.addMove(new XScissor());      }    } |
| package Pokemons;  import ru.ifmo.se.pokemon.Pokemon;  import ru.ifmo.se.pokemon.Type;  import Moves.\*;  public class Snover extends Pokemon {      public Snover(String name, int level) {          super(name, level);          /\*           \* HP               60           \* Attack           62           \* Defense          50           \* Special Attack   62           \* Spacial Defense  60           \* Speed            40           \*/          this.setStats(60d, 62d, 50d, 62d, 60d, 40d);          this.setType(Type.ICE);            this.setMove(new Rest(), new IcyWind(), new EnergyBall());      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class Confide extends StatusMove {      public Confide() {          /\*           \* power -           \* accuracy -           \* priority 0           \* hits 1           \*/          super(Type.NORMAL, 0, 0);      }      /\*       \* Confide lowers the target's Special Attack by one stage.       \*       \* Stats can be lowered to a minimum of -6 stages each.       \*/      @Override      protected void applyOppEffects(Pokemon p) {          p.addEffect(new Effect().stat(Stat.SPECIAL\_ATTACK, -1));      }      @Override      protected String describe() {          return "uses Confide";      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class EnergyBall extends SpecialMove {      public EnergyBall() {          /\*           \* power 90           \* accuracy 100           \* priority 0           \* hits 1           \*/          super(Type.GRASS, 90, 100, 0, 1);      }      /\*       \* Energy Ball deals damage and has a 10% chance of lowering the target's       \* Special Defense by one stage.       \*       \* Stats can be lowered to a minimum of -6 stages each.       \*/      @Override      protected void applyOppEffects(Pokemon p) {          p.addEffect(new Effect().chance(0.1).stat(Stat.SPECIAL\_DEFENSE, -1));      }      @Override      protected String describe() {          return "uses Energy Ball";      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class Extrasensory extends SpecialMove {      public Extrasensory() {          /\*           \* power 80           \* accuracy 100           \* priority 0           \* hits 1           \*/          super(Type.PSYCHIC, 80, 100, 0, 1);      }      /\*       \* Extrasensory deals damage and has a 10% chance of causing the target to       \* flinch (if the target has not yet moved).       \*       \* Pokémon with the ability Inner Focus or those behind a Substitute cannot be       \* made to flinch.       \*/      @Override      protected void applyOppEffects(Pokemon p) {          Effect flinchEffect = new Effect().chance(0.1);          flinchEffect.flinch(p);          p.addEffect(flinchEffect);      }      @Override      protected String describe() {          return "uses Extrasensory";      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class Facade extends PhysicalMove {      public Facade() {          /\*           \* power 70           \* accuracy 100           \* priority 0           \* hits 1           \*/          super(Type.NORMAL, 70, 100, 0, 1);      }      /\*       \* Facade deals damage, and hits with double power (140) if the user is burned,       \* poisoned or paralyzed.       \*       \* In the case of a burn, the usual attack-halving still occurs so Facade hits       \* with an effective power of 70.       \*/      @Override      protected void applyOppEffects(Pokemon p) {          p.setMod(Stat.ATTACK,                  switch (p.getCondition().toString()) {                      case "burned", "poisoned", "paralyzed" -> 2;                      default -> 1;                  });      }      @Override      protected String describe() {          return "uses Facade";      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class IcyWind extends SpecialMove {      public IcyWind() {          /\*           \* power 55           \* accuracy 95           \* priority 0           \* hits 1           \*/          super(Type.ICE, 55, 95, 0, 1);      }      /\*       \* Icy Wind deals damage and lowers the target's Speed by one stage.       \*       \* Stats can be lowered to a minimum of -6 stages each.       \*/      @Override      protected void applyOppEffects(Pokemon p) {          p.addEffect(new Effect().stat(Stat.SPEED, -1));      }      @Override      protected String describe() {          return "uses Icy Wind";      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class Nuzzle extends PhysicalMove {      public Nuzzle() {          /\*           \* power 20           \* accuracy 100           \* priority 0           \* hits 1           \*/          super(Type.ELECTRIC, 20, 100, 0, 1);      }      /\*       \* Nuzzle deals damage and paralyzes the target. Paralyzed Pokémon have a 25%       \* chance of not being able to attack, and their Speed is decreased by 50% (75%       \* in Generations 1-6).       \*       \* Electric type Pokémon, those with the ability Limber or those behind a       \* Substitute cannot be paralyzed.       \*/      @Override      protected void applyOppEffects(Pokemon p) {          p.addEffect(new Effect().condition(Status.PARALYZE));      }      @Override      protected String describe() {          return "uses Nuzzle";      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class Rest extends StatusMove {      public Rest() {          /\*           \* power -           \* accuracy -           \* priority 0           \* hits 1           \*/          super(Type.PSYCHIC, 0, 0);      }      /\*       \* User sleeps for 2 turns, but user is fully healed.       \*/      @Override      protected void applySelfEffects(Pokemon self) {          self.addEffect(new Effect().condition(Status.SLEEP).turns(2));          self.restore();      }      @Override      protected String describe() {          return "uses Rest";      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class RockSlide extends PhysicalMove {      public RockSlide() {          /\*           \* power 75           \* accuracy 90           \* priority 0           \* hits 1           \*/          super(Type.ROCK, 75, 90, 0, 1);      }      /\*       \* Rock Slide deals damage and has a 30% chance of causing the target to flinch       \* (if the target has not yet moved).       \*       \* Pokémon with the ability Inner Focus or those behind a Substitute cannot be       \* made to flinch..       \*/      @Override      protected void applyOppEffects(Pokemon p) {          Effect flinchEffect = new Effect().chance(0.3);          flinchEffect.flinch(p);          p.addEffect(flinchEffect);      }      @Override      protected String describe() {          return "uses Rock Slide";      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class WorkUp extends StatusMove {      public WorkUp() {          /\*           \* power -           \* accuracy -           \* priority 0           \* hits 1           \*/          super(Type.NORMAL, 0, 0);      }      /\*       \* Rock Slide deals damage and has a 30% chance of causing the target to flinch       \* (if the target has not yet moved).       \*       \* Pokémon with the ability Inner Focus or those behind a Substitute cannot be       \* made to flinch.       \*/      @Override      protected void applySelfEffects(Pokemon self) {          self.addEffect(new Effect().stat(Stat.ATTACK, 1).stat(Stat.SPECIAL\_ATTACK, 1));      }      @Override      protected String describe() {          return "uses Work Up";      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class XScissor extends PhysicalMove {      public XScissor() {          /\*           \* power 80           \* accuracy 100           \* priority 0           \* hits 1           \*/          super(Type.BUG, 80, 100, 0, 1);      }      /\*       \* X-Scissor deals damage with no additional effect.       \*/      @Override      protected String describe() {          return "uses X-Scissor";      }  } |
| package Moves;  import ru.ifmo.se.pokemon.\*;  public final class ZingZap extends PhysicalMove {      public ZingZap() {          /\*           \* power     80           \* accuracy  100           \* priority  0           \* hits      1           \*/          super(Type.ELECTRIC, 80, 100, 0, 1);      }      /\*       \* May cause flinching. (30% chance)       \*/      @Override      protected void applyOppEffects(Pokemon p) {          Effect flinchEffect = new Effect().chance(0.3);          flinchEffect.flinch(p);          p.addEffect(flinchEffect);      }      @Override      protected String describe() {          return "uses Zing Zap";      }   } |