```
// This file defines the types and classes used in the Pokemon Battler
application.

/**
 * This class represents a Pokémon with its attributes and methods.
 * The Pokemon class represents a Pokémon with various attributes such as
abilities, stats, moves, and more.
 * It is used to create Pokémon objects that can be used for a proper presentation
of a Pokémon in the Team Builder part of the application.
 * It includes methods for cloning and preparing the Pokémon for battle.
 * @param pdx_num - The Pokémon's Pokédex number.
 * @param name - The Pokémon's name.
 * @param ability - The Pokémon's ability, which can be null if it has no set
ability.
 * @param abilitys - An array of the Pokémon's possible abilities.
 * @param lvl - The Pokémon's level, which must be between 1 and 100.
 * @param gender - The Pokémon's gender.
 * @param nature - The Pokémon's nature, which affects its stats.
 * @param shiny - Whether the Pokémon is shiny or not.
 * @param types - An array of the Pokémon's types.
 * @param ivs - An array of the Pokémon's individual values (IVs).
 * @param evs - An array of the Pokémon's effort values (EVs).
 * @param stats - An array of the Pokémon's base stats.
 * @param moves - An array of the Pokémon's moves.
 * @param moveset - An array representing the Pokémon's current moveset.
 * @param sprite - The URL of the Pokémon's front sprite.
 * @param sprite_back - The URL of the Pokémon's back sprite.
 * @function clone() - Creates a deep copy of the Pokémon instance.
 * @function makeBattleReady() - Prepares the Pokémon for battle by ensuring it
has a valid moveset.
 */
export class Pokemon {
    pdx_num: number
    name: string
    ability: Ability | null
    abilitys: Ability[]
    lvl = 100
    gender = "Male"
    nature = "Hardy"
    shiny = false
    types: string[]
    ivs: Ivs[] = [
        new Ivs("hp", 31),
        new Ivs("attack", 31),
        new Ivs("defense", 31),
        new Ivs("special-attack", 31),
        new Ivs("special-defense", 31),
        new Ivs("speed", 31),
    ]

    evs: Evs[] = [
        new Evs("hp", 0),
```

```
            new Evs("attack", 0),
            new Evs("defense", 0),
            new Evs("special-attack", 0),
            new Evs("special-defense", 0),
            new Evs("speed", 0),
        ]
        stats: Stats[]
        moves: Moves[]
        moveset: (Moves | null)[] = [null, null, null, null]
        sprite: string
        sprite_back: string

        constructor(
            pdx_num: number,
            name: string,
            ability: Ability | null,
            abilitys: Ability[],
            lvl: number,
            gender: string,
            nature: string,
            shiny: boolean,
            types: string[],
            ivs: Ivs[],
            evs: Evs[],
            stats: Stats[],
            moves: Moves[],
            moveset: (Moves | null)[],
            sprite: string,
            sprite_back: string,
        ) {
            this.pdx_num = pdx_num
            this.name = name
            this.ability = ability
            this.abilitys = abilitys
            if (lvl < 1 || lvl > 100) {
                throw new Error(`Invalid level: ${lvl}`)
            }
            this.lvl = lvl
            this.gender = gender
            if (!Object.values(Natures).includes(nature as Natures)) {
                throw new Error(`Invalid Nature: ${nature}`)
            }
            this.nature = nature
            this.shiny = shiny
            this.types = types
            this.stats = stats
            this.ivs = ivs
            this.evs = evs
            this.moves = moves
            this.moveset = [...moveset]
            this.sprite = sprite
            this.sprite_back = sprite_back
        }
```

```
    /*Getter*/
    getPdx_num(): number {
        return this.pdx_num
    }

    getName(): string {
        return this.name
    }

    getAbility(): Ability | null {
        return this.ability
    }

    getAbilitys(): Ability[] {
        return this.abilitys
    }
    getLvl(): number {
        return this.lvl
    }

    getGender(): string {
        return this.gender
    }

    getNature(): string {
        return this.nature
    }

    getShiny(): boolean {
        return this.shiny
    }

    getTypes(): string[] {
        return this.types
    }

    getStats(): Stats[] {
        return this.stats
    }

    getMoves(): Moves[] {
        return this.moves
    }

    getMoveset(): (Moves | null)[] {
        return [...this.moveset]
    }

    getSprite(): string {
        return this.sprite
    }

    getSprite_back(): string {
        return this.sprite_back
```

```
    }

    /*Setter*/
    setPdx_num(pdx_num: number) {
        this.pdx_num = pdx_num
    }
    setName(name: string) {
        if (!name || name.trim() === "") {
            throw new Error("Name cannot be empty")
        }
        this.name = name
    }
    setTypes(types: string[]) {
        this.types = types
    }
    setLvl(lvl: number) {
        if (lvl < 1 || lvl > 100) {
            throw new Error(`Invalid level: ${lvl}`)
        }
        this.lvl = lvl
    }
    setGender(gender: string) {
        this.gender = gender
    }
    setNature(nature: string) {
        if (!Object.values(Natures).includes(nature as Natures)) {
            throw new Error(`Invalid Nature: ${nature}`)
        }
        this.nature = nature
    }
    setShiny(shiny: boolean) {
        this.shiny = shiny
    }
    setStats(stats: Stats[]) {
        for (let i = 0; i < 6; i++) {
            if (stats[i].basestat < 0) {
                throw new Error(`Invalid stat: ${stats[i]}`)
            }
            // ☑ Nur ersetzen, nicht erneut addieren
            this.stats[i].basestat = stats[i].basestat
        }
    }
    setMoveset(index: number, move: Moves | null) {
        if (index >= 0 && index < 4) {
            this.moveset[index] = move
        }
    }
    setMoves(moves: Moves[]) {
        this.moves = moves
    }
    setIvs(ivs: Ivs[]) {
        for (let i = 0; i < 6; i++) {
            if (ivs[i].value < 0 || ivs[i].value > 31) {
```

```
                throw new Error(`Invalid IV: ${ivs[i]}`)
            }
            this.ivs[i].value = ivs[i].value
        }
    }
    setEvs(evs: Evs[]) {
        for (let i = 0; i < 6; i++) {
            if (evs[i].value < 0 || evs[i].value > 252) {
                throw new Error(`Invalid EV: ${evs[i]}`)
            }
            this.evs[i].value = evs[i].value
        }
    }
    setAbility(ability: Ability | null) {
        this.ability = ability
    }
    setAbilitys(abilitys: Ability[]) {
        this.abilitys = abilitys
    }
    setSprite(sprite: string) {
        this.sprite = sprite
    }
    setSprite_back(sprite_back: string) {
        this.sprite_back = sprite_back
    }

    /**
     * This function creates a deep copy of the current instance of Pokemon.
     * It ensures that all properties are copied correctly, including nested
objects like Ability, Ivs, Evs, Stats, Moves, and Moveset.
     * @returns A new instance of Pokemon with the same values as the current
instance.
     */
    clone(): Pokemon {
        // Deep copy von Ability (null-safe)
        const clonedAbility = this.ability ? new Ability(this.ability.name,
this.ability.effect) : null

        // Deep copy der Abilitys
        const clonedAbilitys = this.abilitys.map((ab) => new Ability(ab.name,
ab.effect))

        // Deep copy IVs
        const clonedIvs = this.ivs.map((iv) => new Ivs(iv.name, iv.value))

        // Deep copy EVs
        const clonedEvs = this.evs.map((ev) => new Evs(ev.name, ev.value))

        // Deep copy Stats
        const clonedStats = this.stats.map((stat) => new Stats(stat.name,
stat.basestat))

        // Deep copy Moves
        const clonedMoves = this.moves.map(
```

```
            (move) => new Moves(move.name, move.type, move.power, move.accuracy,
move.pp, move.damageClass),
        )

        // Deep copy Moveset (null-safe)
        const clonedMoveset = this.moveset.map((move) =>
            move ? new Moves(move.name, move.type, move.power, move.accuracy,
move.pp, move.damageClass) : null,
        )

        // Rückgabe eines neuen Pokemon-Objekts mit denselben Werten
        return new Pokemon(
            this.pdx_num,
            this.name,
            clonedAbility,
            clonedAbilitys,
            this.lvl,
            this.gender,
            this.nature,
            this.shiny,
            [...this.types],
            clonedIvs,
            clonedEvs,
            clonedStats,
            clonedMoves,
            clonedMoveset,
            this.sprite,
            this.sprite_back,
        )
    }

    /**
     * This function prepares the Pokémon for battle by ensuring it has a valid
moveset.
     * If the moveset is empty, it falls back to the first four moves available.
     * It also calculates the Pokémon's battle stats based on its base stats, IVs,
EVs, and level.
     * @returns A new instance of Pokemon_in_battle with the calculated battle-
ready stats and moveset.
     */
    makeBattleReady(): Pokemon_in_battle {
        const validMoveset = this.getMoveset().filter(move => move !== null) as
Moves[];

        const fallbackMoves = this.getMoves().slice(0, 4); // Falls Moveset leer
ist

        const finalMoveset = validMoveset.length > 0 ? validMoveset :
fallbackMoves;

        // All Base Stats are calculated based on the formula for Pokémon stats:
        // HP = ((2 * Base HP + IVs + (EVs / 4) + 100 * Level) / 100) + Level + 10
        // Other stats are calculated as follows:
        // Stat = ((2 * Base Stat + IVs + (EVs / 4) * Level) / 100) + 5
```

```
        const lvl = this.getLvl();
        const baseHp = this.getStats().find(stat => stat.getName() ===
"hp")?.getBasestat() || 0;
        const baseAttack = this.getStats().find(stat => stat.getName() ===
"attack")?.getBasestat() || 0;
        const baseDefense = this.getStats().find(stat => stat.getName() ===
"defense")?.getBasestat() || 0;
        const baseSpecialAttack = this.getStats().find(stat => stat.getName() ===
"special-attack")?.getBasestat() || 0;
        const baseSpecialDefense = this.getStats().find(stat => stat.getName() ===
"special-defense")?.getBasestat() || 0;
        const baseSpeed = this.getStats().find(stat => stat.getName() ===
"speed")?.getBasestat() || 0;

        const hpIvs = this.ivs.find(iv => iv.getName() === "hp")?.getValue() || 0;
        const hpEvs = this.evs.find(ev => ev.getName() === "hp")?.getValue() || 0;
        const attackIvs = this.ivs.find(iv => iv.getName() ===
"attack")?.getValue() || 0;
        const attackEvs = this.evs.find(ev => ev.getName() ===
"attack")?.getValue() || 0;
        const defenseIvs = this.ivs.find(iv => iv.getName() ===
"defense")?.getValue() || 0;
        const defenseEvs = this.evs.find(ev => ev.getName() ===
"defense")?.getValue() || 0;
        const specialAttackIvs = this.ivs.find(iv => iv.getName() === "special-
attack")?.getValue() || 0;
        const specialAttackEvs = this.evs.find(ev => ev.getName() === "special-
attack")?.getValue() || 0;
        const specialDefenseIvs = this.ivs.find(iv => iv.getName() === "special-
defense")?.getValue() || 0;
        const specialDefenseEvs = this.evs.find(ev => ev.getName() === "special-
defense")?.getValue() || 0;
        const speedIvs = this.ivs.find(iv => iv.getName() === "speed")?.getValue()
|| 0;
        const speedEvs = this.evs.find(ev => ev.getName() === "speed")?.getValue()
|| 0;

        const battleHp = Math.floor(((2 * baseHp + hpIvs + (hpEvs / 4)) * lvl) /
100) + lvl + 10;
        const battleAttack = Math.floor(((2 * baseAttack + attackIvs + (attackEvs
/ 4)) * lvl) / 100) + 5;
        const battleDefense = Math.floor(((2 * baseDefense + defenseIvs +
(defenseEvs / 4)) * lvl) / 100) + 5;
        const battleSpecialAttack = Math.floor(((2 * baseSpecialAttack +
specialAttackIvs + (specialAttackEvs / 4)) * lvl) / 100) + 5;
        const battleSpecialDefense = Math.floor(((2 * baseSpecialDefense +
specialDefenseIvs + (specialDefenseEvs / 4)) * lvl) / 100) + 5;
        const battleSpeed = Math.floor(((2 * baseSpeed + speedIvs + (speedEvs /
4)) * lvl) / 100) + 5;

        return new Pokemon_in_battle(
            this.getPdx_num(),
            this.getName(),
            this.getAbility(),
```

```
            this.getLvl(),
            this.getGender(),
            this.getNature(),
            this.getTypes(),
            battleHp,
            battleHp,
            battleAttack,
            battleDefense,
            battleSpecialAttack,
            battleSpecialDefense,
            battleSpeed,
            finalMoveset,
            this.getSprite(),
            this.getSprite_back(),
        )
    }
}

/**
 * This class represents a Pokémon in battle with its current state and
attributes.
 * It includes properties like current HP, max HP, stats, moveset, and sprites.
 * It uses the crypto library to generate a unique ID for each instance, so that
each Pokémon in battle can be uniquely identified.
 * This class is used to manage the Pokémon's state during battles, including
current HP, moves, and other battle-related attributes.
 * * @param id - A unique identifier for the Pokémon in battle, generated using
crypto.randomUUID().
 * @param pdx_num - The Pokémon's Pokédex number.
 * @param name - The Pokémon's name.
 * @param ability - The Pokémon's ability, which can be null if it has no set
ability.
 * @param lvl - The Pokémon's level, which must be between 1 and 100.
 * @param gender - The Pokémon's gender, which can be "Male", "Female", or
"Genderless".
 * @param nature - The Pokémon's nature, which affects its stat growth.
 * @param types - An array of the Pokémon's types.
 * @param currentHP - The Pokémon's current HP, which must be between 0 and maxHP.
 * @param maxHP - The Pokémon's maximum HP, which must be greater than 0.
 * @param attack - The Pokémon's attack stat.
 * @param defense - The Pokémon's defense stat.
 * @param special_attack - The Pokémon's special attack stat.
 * @param special_defense - The Pokémon's special defense stat.
 * @param speed - The Pokémon's speed stat.
 * @param moveset - An array representing the Pokémon's current moveset.
 * @param sprite - The URL of the Pokémon's front sprite.
 * @param sprite_back - The URL of the Pokémon's back sprite.
 * @function clone() - Creates a deep copy of the Pokémon in battle instance.
 */
export class Pokemon_in_battle {
    id: string = crypto.randomUUID()
    pdx_num: number
    name: string
    ability: Ability | null
```

```
    lvl = 100
    gender = "Male"
    nature = "Hardy"
    types: string[]
    currentHP: number;
    maxHP: number;
    attack: number;
    defense: number;
    special_attack: number;
    special_defense: number;
    speed: number;
    moveset: Moves[] = [];
    sprite: string;
    sprite_back: string;
    constructor(
        pdx_num: number,
        name: string,
        ability: Ability | null,
        lvl: number,
        gender: string,
        nature: string,
        types: string[],
        currentHP: number,
        maxHP: number,
        attack: number,
        defense: number,
        special_attack: number,
        special_defense: number,
        speed: number,
        moveset: (Moves)[],
        sprite: string,
        sprite_back: string,
    ) {
        this.pdx_num = pdx_num
        this.name = name
        this.ability = ability
        this.lvl = lvl
        this.gender = gender
        this.nature = nature
        this.types = types
        this.currentHP = currentHP
        this.maxHP = maxHP
        this.attack = attack
        this.defense = defense
        this.special_attack = special_attack
        this.special_defense = special_defense
        this.speed = speed
        this.moveset = moveset
        this.sprite = sprite
        this.sprite_back = sprite_back

    }
    /*Getter*/
    getId(): string {
```

```
        return this.id
    }
    getPdx_num(): number {
        return this.pdx_num
    }
    getName(): string {
        return this.name
    }
    getAbility(): Ability | null {
        return this.ability
    }
    getLvl(): number {
        return this.lvl
    }
    getGender(): string {
        return this.gender
    }
    getNature(): string {
        return this.nature
    }
    getTypes(): string[] {
        return this.types
    }
    getCurrentHP(): number {
        return this.currentHP
    }
    getMaxHP(): number {
        return this.maxHP
    }
    getAttack(): number {
        return this.attack
    }
    getDefense(): number {
        return this.defense
    }
    getSpecialAttack(): number {
        return this.special_attack
    }
    getSpecialDefense(): number {
        return this.special_defense
    }
    getSpeed(): number {
        return this.speed
    }
    getMoveset(): Moves[] {
        return this.moveset
    }
    getSprite(): string {
        return this.sprite
    }
    getSprite_back(): string {
        return this.sprite_back

    }
```

```
    /*Setter*/
    setCurrentHP(currentHP: number) {
        if (currentHP < 0 || currentHP > this.maxHP) {
            throw new Error(`Invalid current HP: ${currentHP}`)
        }
        this.currentHP = currentHP
    }
    setCurrentPP(moveName: string, pp: number) {
        const move = this.moveset.find(m => m.name === moveName)
        if (move) {
            move.current_pp = pp
        }
    }
    getCurrentPP(moveName: string): number {
        const move = this.moveset.find(m => m.name === moveName)
        if (move) {
            return move.current_pp
        }
        throw new Error(`Move not found: ${moveName}`)
    }

    /**
     * This function creates a deep copy of the current instance of
Pokemon_in_battle.
     * It ensures that all properties are copied correctly, including nested
objects like Ability and Moves.
     * *It is used to create a new instance of Pokemon_in_battle with the same
values as the current instance,
     * so that the original instance remains unchanged.
     * @returns A new instance of Pokemon_in_battle with the same values as the
current instance.
     */
    clone(): Pokemon_in_battle {
        // Deep copy von Ability (null-safe)
        const clonedAbility = this.ability ? new Ability(this.ability.name,
this.ability.effect) : null

        // Deep copy Moves
        const clonedMoveset = this.moveset.map(
            (move) => new Moves(move.name, move.type, move.power, move.accuracy,
move.pp, move.damageClass, move.current_pp),
        )

        // return a new instance of Pokemon_in_battle with the same values
        // This ensures that the cloned instance has its own unique ID and does
not share references with the original instance.
        return new Pokemon_in_battle(
            this.pdx_num,
            this.name,
            clonedAbility,
            this.lvl,
            this.gender,
            this.nature,
            this.types,
```

```
            this.currentHP,
            this.maxHP,
            this.attack,
            this.defense,
            this.special_attack,
            this.special_defense,
            this.speed,
            clonedMoveset,
            this.sprite,
            this.sprite_back,
        )
    }
}


/**
 * This class represents a Pokémon's ability with its name and effect.
 * Abilities can have various effects, such as boosting stats or providing
immunities.
 * @param name - The name of the ability.
 * @param effect - The effect of the ability, which describes what it does in
battle.
 */
export class Ability {
    name: string
    effect: string

    constructor(name: string, effect: string) {
        this.name = name
        this.effect = effect
    }
    /*Getter*/
    getName(): string {
        return this.name
    }

    getEffect(): string {
        return this.effect
    }
}

/**
 * This class represents a Pokémon's stat with its name and base stat value.
 * @param name - The name of the stat (e.g., "hp", "attack", "defense").
 * @param basestat - The base value of the stat, which is used to calculate the
Pokémon's actual stat in battle.
 */
export class Stats {
    name: string
    basestat: number

    constructor(name: string, basestat: number) {
        this.name = name
        this.basestat = basestat
```

```
    }

    getName(): string {
        return this.name
    }

    getBasestat(): number {
        return this.basestat
    }
}

/**
 * This class represents a Pokémon's individual values (IVs) for each stat.
 * IVs are hidden stats that determine how good a Pokémon can potentially be.
 * They range from 0 to 31 for each stat, with higher values indicating better
potential.
 * @param name - The name of the stat (e.g., "hp", "attack", "defense").
 * @param value - The IV value for the stat, which must be between 0 and 31.
 */
export class Ivs {
    name: string
    value: number
    constructor(name: string, value: number) {
        this.name = name
        this.value = value
    }
    getName(): string {
        return this.name
    }
    getValue(): number {
        return this.value
    }
    setValue(value: number) {
        if (value < 0 || value > 31) {
            throw new Error(`Invalid IV: ${value}`)
        }
        this.value = value
    }
}


/**
 * This class represents a Pokémon's effort values (EVs) for each stat.
 * EVs are points that can be allocated to a Pokémon's stats to enhance their
performance in battles.
 * They range from 0 to 252 for each stat, with a maximum total of 510 EVs across
all stats.
 * @param name - The name of the stat (e.g., "hp", "attack", "defense").
 * @param value - The EV value for the stat, which must be between 0 and 252.
 */
export class Evs {
    name: string
    value: number
    constructor(name: string, value: number) {
```

```
        this.name = name
        this.value = value
    }
    getName(): string {
        return this.name
    }
    getValue(): number {
        return this.value
    }
    setValue(value: number) {
        if (value < 0 || value > 252) {
            throw new Error(`Invalid EV: ${value}`)
        }
        this.value = value
    }
}

/**
 * This class represents a Pokémon's move, with its name, type, power, accuracy,
PP (Power Points), and damage class.
 * Moves are the actions that Pokémon can perform in battles, and they can vary in
their effects and attributes.
 * @param name - The name of the move.
 * @param type - The type of the move (e.g., "fire", "water", "grass").
 * @param power - The power of the move, which determines its damage output.
 * @param accuracy - The accuracy of the move, which determines how likely it is
to hit the target.
 * @param pp - The maximum Power Points (PP) for the move, which determines how
many times it can be used in battle.
 * @param damageClass - The damage class of the move (e.g., "physical", "special",
"status").
 * @param current_pp - The current Power Points (PP) for the move, which can be
less than or equal to pp.
 */
export class Moves {
    name: string
    type: string
    power: number
    accuracy: number
    current_pp: number
    pp: number
    damageClass: string

    constructor(name: string, type: string, power: number, accuracy: number, pp:
number, damageClass: string, current_pp: number = pp) {
        this.name = name
        this.type = type
        this.power = power
        this.accuracy = accuracy
        this.pp = pp
        this.current_pp = current_pp ? current_pp : pp
        this.damageClass = damageClass
    }
}
```

```ts
// This object defines the available natures for Pokémon.
// Each nature affects the Pokémon's stats in a specific way, such as increasing one stat while decreasing another.
// Currently Natures aren't used in the application, but they can be used in the future to enhance the Pokémon's attributes and gameplay mechanics.
export const Natures = {
    hardy: "Hardy",
    lonely: "Lonely",
    brave: "Brave",
    adamant: "Adamant",
    naughty: "Naughty",
    bold: "Bold",
    relaxed: "Relaxed",
    impish: "Impish",
    lax: "Lax",
    timid: "Timid",
    hasty: "Hasty",
    jolly: "Jolly",
    naive: "Naive",
    modest: "Modest",
    mild: "Mild",
    quiet: "Quiet",
    rash: "Rash",
    calm: "Calm",
    gentle: "Gentle",
    sassy: "Sassy",
} as const

export type Natures = (typeof Natures)[keyof typeof Natures]
```