export interface CollectionConstructor {

new (): Collection<unknown, unknown>;

new <K, V>(entries?: ReadonlyArray<readonly [K, V]> | null): Collection<K, V>;

new <K, V>(iterable: Iterable<readonly [K, V]>): Collection<K, V>;

readonly prototype: Collection<unknown, unknown>;

readonly [Symbol.species]: CollectionConstructor;

}

/\*\*

\* A Map with additional utility methods. This is used throughout discord.js rather than Arrays for anything that has

\* an ID, for significantly improved performance and ease-of-use.

\* @extends {Map}

\* @property {number} size - The amount of elements in this collection.

\*/

declare class Collection<K, V> extends Map<K, V> {

private \_array;

private \_keyArray;

static readonly default: typeof Collection;

['constructor']: typeof Collection;

constructor(entries?: ReadonlyArray<readonly [K, V]> | null);

/\*\*

\* Identical to [Map.get()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map/get).

\* Gets an element with the specified key, and returns its value, or `undefined` if the element does not exist.

\* @param {\*} key - The key to get from this collection

\* @returns {\* | undefined}

\*/

get(key: K): V | undefined;

/\*\*

\* Identical to [Map.set()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map/set).

\* Sets a new element in the collection with the specified key and value.

\* @param {\*} key - The key of the element to add

\* @param {\*} value - The value of the element to add

\* @returns {Collection}

\*/

set(key: K, value: V): this;

/\*\*

\* Identical to [Map.has()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map/has).

\* Checks if an element exists in the collection.

\* @param {\*} key - The key of the element to check for

\* @returns {boolean} `true` if the element exists, `false` if it does not exist.

\*/

has(key: K): boolean;

/\*\*

\* Identical to [Map.delete()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map/delete).

\* Deletes an element from the collection.

\* @param {\*} key - The key to delete from the collection

\* @returns {boolean} `true` if the element was removed, `false` if the element does not exist.

\*/

delete(key: K): boolean;

/\*\*

\* Identical to [Map.clear()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map/clear).

\* Removes all elements from the collection.

\* @returns {undefined}

\*/

clear(): void;

/\*\*

\* Creates an ordered array of the values of this collection, and caches it internally. The array will only be

\* reconstructed if an item is added to or removed from the collection, or if you change the length of the array

\* itself. If you don't want this caching behavior, use `[...collection.values()]` or

\* `Array.from(collection.values())` instead.

\* @returns {Array}

\*/

array(): V[];

/\*\*

\* Creates an ordered array of the keys of this collection, and caches it internally. The array will only be

\* reconstructed if an item is added to or removed from the collection, or if you change the length of the array

\* itself. If you don't want this caching behavior, use `[...collection.keys()]` or

\* `Array.from(collection.keys())` instead.

\* @returns {Array}

\*/

keyArray(): K[];

/\*\*

\* Obtains the first value(s) in this collection.

\* @param {number} [amount] Amount of values to obtain from the beginning

\* @returns {\*|Array<\*>} A single value if no amount is provided or an array of values, starting from the end if

\* amount is negative

\*/

first(): V | undefined;

first(amount: number): V[];

/\*\*

\* Obtains the first key(s) in this collection.

\* @param {number} [amount] Amount of keys to obtain from the beginning

\* @returns {\*|Array<\*>} A single key if no amount is provided or an array of keys, starting from the end if

\* amount is negative

\*/

firstKey(): K | undefined;

firstKey(amount: number): K[];

/\*\*

\* Obtains the last value(s) in this collection. This relies on {@link Collection#array}, and thus the caching

\* mechanism applies here as well.

\* @param {number} [amount] Amount of values to obtain from the end

\* @returns {\*|Array<\*>} A single value if no amount is provided or an array of values, starting from the start if

\* amount is negative

\*/

last(): V | undefined;

last(amount: number): V[];

/\*\*

\* Obtains the last key(s) in this collection. This relies on {@link Collection#keyArray}, and thus the caching

\* mechanism applies here as well.

\* @param {number} [amount] Amount of keys to obtain from the end

\* @returns {\*|Array<\*>} A single key if no amount is provided or an array of keys, starting from the start if

\* amount is negative

\*/

lastKey(): K | undefined;

lastKey(amount: number): K[];

/\*\*

\* Obtains unique random value(s) from this collection. This relies on {@link Collection#array}, and thus the caching

\* mechanism applies here as well.

\* @param {number} [amount] Amount of values to obtain randomly

\* @returns {\*|Array<\*>} A single value if no amount is provided or an array of values

\*/

random(): V;

random(amount: number): V[];

/\*\*

\* Obtains unique random key(s) from this collection. This relies on {@link Collection#keyArray}, and thus the caching

\* mechanism applies here as well.

\* @param {number} [amount] Amount of keys to obtain randomly

\* @returns {\*|Array<\*>} A single key if no amount is provided or an array

\*/

randomKey(): K;

randomKey(amount: number): K[];

/\*\*

\* Searches for a single item where the given function returns a truthy value. This behaves like

\* [Array.find()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/find).

\* <warn>All collections used in Discord.js are mapped using their `id` property, and if you want to find by id you

\* should use the `get` method. See

\* [MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map/get) for details.</warn>

\* @param {Function} fn The function to test with (should return boolean)

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {\*}

\* @example collection.find(user => user.username === 'Bob');

\*/

find(fn: (value: V, key: K, collection: this) => boolean): V | undefined;

find<T>(fn: (this: T, value: V, key: K, collection: this) => boolean, thisArg: T): V | undefined;

/\*\*

\* Searches for the key of a single item where the given function returns a truthy value. This behaves like

\* [Array.findIndex()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/findIndex),

\* but returns the key rather than the positional index.

\* @param {Function} fn The function to test with (should return boolean)

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {\*}

\* @example collection.findKey(user => user.username === 'Bob');

\*/

findKey(fn: (value: V, key: K, collection: this) => boolean): K | undefined;

findKey<T>(fn: (this: T, value: V, key: K, collection: this) => boolean, thisArg: T): K | undefined;

/\*\*

\* Removes items that satisfy the provided filter function.

\* @param {Function} fn Function used to test (should return a boolean)

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {number} The number of removed entries

\*/

sweep(fn: (value: V, key: K, collection: this) => boolean): number;

sweep<T>(fn: (this: T, value: V, key: K, collection: this) => boolean, thisArg: T): number;

/\*\*

\* Identical to

\* [Array.filter()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/filter),

\* but returns a Collection instead of an Array.

\* @param {Function} fn The function to test with (should return boolean)

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {Collection}

\* @example collection.filter(user => user.username === 'Bob');

\*/

filter(fn: (value: V, key: K, collection: this) => boolean): this;

filter<T>(fn: (this: T, value: V, key: K, collection: this) => boolean, thisArg: T): this;

/\*\*

\* Partitions the collection into two collections where the first collection

\* contains the items that passed and the second contains the items that failed.

\* @param {Function} fn Function used to test (should return a boolean)

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {Collection[]}

\* @example const [big, small] = collection.partition(guild => guild.memberCount > 250);

\*/

partition(fn: (value: V, key: K, collection: this) => boolean): [this, this];

partition<T>(fn: (this: T, value: V, key: K, collection: this) => boolean, thisArg: T): [this, this];

/\*\*

\* Maps each item into a Collection, then joins the results into a single Collection. Identical in behavior to

\* [Array.flatMap()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/flatMap).

\* @param {Function} fn Function that produces a new Collection

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {Collection}

\* @example collection.flatMap(guild => guild.members.cache);

\*/

flatMap<T>(fn: (value: V, key: K, collection: this) => Collection<K, T>): Collection<K, T>;

flatMap<T, This>(fn: (this: This, value: V, key: K, collection: this) => Collection<K, T>, thisArg: This): Collection<K, T>;

/\*\*

\* Maps each item to another value into an array. Identical in behavior to

\* [Array.map()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/map).

\* @param {Function} fn Function that produces an element of the new array, taking three arguments

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {Array}

\* @example collection.map(user => user.tag);

\*/

map<T>(fn: (value: V, key: K, collection: this) => T): T[];

map<This, T>(fn: (this: This, value: V, key: K, collection: this) => T, thisArg: This): T[];

/\*\*

\* Maps each item to another value into a collection. Identical in behavior to

\* [Array.map()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/map).

\* @param {Function} fn Function that produces an element of the new collection, taking three arguments

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {Collection}

\* @example collection.mapValues(user => user.tag);

\*/

mapValues<T>(fn: (value: V, key: K, collection: this) => T): Collection<K, T>;

mapValues<This, T>(fn: (this: This, value: V, key: K, collection: this) => T, thisArg: This): Collection<K, T>;

/\*\*

\* Checks if there exists an item that passes a test. Identical in behavior to

\* [Array.some()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/some).

\* @param {Function} fn Function used to test (should return a boolean)

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {boolean}

\* @example collection.some(user => user.discriminator === '0000');

\*/

some(fn: (value: V, key: K, collection: this) => boolean): boolean;

some<T>(fn: (this: T, value: V, key: K, collection: this) => boolean, thisArg: T): boolean;

/\*\*

\* Checks if all items passes a test. Identical in behavior to

\* [Array.every()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/every).

\* @param {Function} fn Function used to test (should return a boolean)

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {boolean}

\* @example collection.every(user => !user.bot);

\*/

every(fn: (value: V, key: K, collection: this) => boolean): boolean;

every<T>(fn: (this: T, value: V, key: K, collection: this) => boolean, thisArg: T): boolean;

/\*\*

\* Applies a function to produce a single value. Identical in behavior to

\* [Array.reduce()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/reduce).

\* @param {Function} fn Function used to reduce, taking four arguments; `accumulator`, `currentValue`, `currentKey`,

\* and `collection`

\* @param {\*} [initialValue] Starting value for the accumulator

\* @returns {\*}

\* @example collection.reduce((acc, guild) => acc + guild.memberCount, 0);

\*/

reduce<T>(fn: (accumulator: T, value: V, key: K, collection: this) => T, initialValue?: T): T;

/\*\*

\* Identical to

\* [Map.forEach()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map/forEach),

\* but returns the collection instead of undefined.

\* @param {Function} fn Function to execute for each element

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {Collection}

\* @example

\* collection

\* .each(user => console.log(user.username))

\* .filter(user => user.bot)

\* .each(user => console.log(user.username));

\*/

each(fn: (value: V, key: K, collection: this) => void): this;

each<T>(fn: (this: T, value: V, key: K, collection: this) => void, thisArg: T): this;

/\*\*

\* Runs a function on the collection and returns the collection.

\* @param {Function} fn Function to execute

\* @param {\*} [thisArg] Value to use as `this` when executing function

\* @returns {Collection}

\* @example

\* collection

\* .tap(coll => console.log(coll.size))

\* .filter(user => user.bot)

\* .tap(coll => console.log(coll.size))

\*/

tap(fn: (collection: this) => void): this;

tap<T>(fn: (this: T, collection: this) => void, thisArg: T): this;

/\*\*

\* Creates an identical shallow copy of this collection.

\* @returns {Collection}

\* @example const newColl = someColl.clone();

\*/

clone(): this;

/\*\*

\* Combines this collection with others into a new collection. None of the source collections are modified.

\* @param {...Collection} collections Collections to merge

\* @returns {Collection}

\* @example const newColl = someColl.concat(someOtherColl, anotherColl, ohBoyAColl);

\*/

concat(...collections: Collection<K, V>[]): this;

/\*\*

\* Checks if this collection shares identical items with another.

\* This is different to checking for equality using equal-signs, because

\* the collections may be different objects, but contain the same data.

\* @param {Collection} collection Collection to compare with

\* @returns {boolean} Whether the collections have identical contents

\*/

equals(collection: Collection<K, V>): boolean;

/\*\*

\* The sort method sorts the items of a collection in place and returns it.

\* The sort is not necessarily stable in Node 10 or older.

\* The default sort order is according to string Unicode code points.

\* @param {Function} [compareFunction] Specifies a function that defines the sort order.

\* If omitted, the collection is sorted according to each character's Unicode code point value,

\* according to the string conversion of each element.

\* @returns {Collection}

\* @example collection.sort((userA, userB) => userA.createdTimestamp - userB.createdTimestamp);

\*/

sort(compareFunction?: (firstValue: V, secondValue: V, firstKey: K, secondKey: K) => number): this;

/\*\*

\* The intersect method returns a new structure containing items where the keys are present in both original structures.

\* @param {Collection} other The other Collection to filter against

\* @returns {Collection}

\*/

intersect(other: Collection<K, V>): Collection<K, V>;

/\*\*

\* The difference method returns a new structure containing items where the key is present in one of the original structures but not the other.

\* @param {Collection} other The other Collection to filter against

\* @returns {Collection}

\*/

difference(other: Collection<K, V>): Collection<K, V>;

/\*\*

\* The sorted method sorts the items of a collection and returns it.

\* The sort is not necessarily stable in Node 10 or older.

\* The default sort order is according to string Unicode code points.

\* @param {Function} [compareFunction] Specifies a function that defines the sort order.

\* If omitted, the collection is sorted according to each character's Unicode code point value,

\* according to the string conversion of each element.

\* @returns {Collection}

\* @example collection.sorted((userA, userB) => userA.createdTimestamp - userB.createdTimestamp);

\*/

sorted(compareFunction?: (firstValue: V, secondValue: V, firstKey: K, secondKey: K) => number): this;

}

export { Collection };

export default Collection;