Map

- **get(key)**
  - return the value against key; if not present then null
- **getOrDefault(key, default_value)**
  - if key not present, return default
- **put(key, value)**
  - stores/replaces the value
- **putIfAbsent(key, value)**
  - if key is not present then put; else ignore
- **containsKey(key)**
  - returns true/false if key present/absent
- **remove(key)**
  - remove the key
- **merge(key, val, (old_val, new_val) -> result)**
  - if new value against a key is dependent upon the previous value
  - val will be passed as new_val to lambda function if key already exists
  - val will be inserted against the key key if key is not present; in this case the lambda function will not be called.
  -
```
// merge: append new value to old value with space
mp.merge("name", "Ranjan", (oldValue, newValue) -> oldValue + " " + newValue);

System.out.println(mp);  // {name=Ranjan}

// Now add another value for "name"
mp.merge("name", "Alok", (oldValue, newValue) -> oldValue + " " + newValue);

System.out.println(mp);  // {name=Ranjan Alok}
```
- **compute(key, (k, v) -> result)**
  - do some operation to the previously present value
  - mp.compute("count", (k, v) -> v == null ? 1 : v+1)
  - If count is there then increase the value by 1, else insert 1.
- **entrySet()**
  - returns the Set of Entry type objects.
  - Set<Map.Entry<K, V>> entrySet() Entry is inner interface inside Map

```
for (Map.Entry<Integer, Integer> e : map.entrySet()) {
    System.out.println(e.getKey() + " " + e.getValue());
}
```

- **keySet(), valueSet()**
  - returns Set of key and values respectively.
  - ```
    for (int key : map.keySet()) { ... }
    ```
  - ```
    for (int v : map.values()) { ... }
    ```
- NOTE
  - **compute()** and **merge()** looks similar. But the difference is:
    - In lambda function of **compute** : **key** and **value** is passed
    - In lambda function of **merge** : **old_value** and **new_value** is passed
- F

List

- F
- F
- F
- F
- F
-