

Functional programming

- A function is a first class object (in languages that support functional programming)
- You can pass a function as an argument of a function
- You can return a function from another function
- You can assign a function to a variable
 - not in Java

Functional interface

- An interface with exactly 1 abstract method
- Can also be annotated with `@FunctionalInterface`
- In Java, we can assign an object of any interface to a variable
 - Since in a functional interface, there is exactly 1 abstract function signature, we can assign an implementation for that function to a variable of a functional interface

For example, the interface called `java.lang.Runnable` has one abstract method `public void run()`.

```
Runnable r1 = new Runnable() {  
    public void run(){  
        System.out.println("Hello");  
    }  
};  
  
// idea is  
Runnable r2 = public void run(){  
    System.out.println("Hello");  
};  
// should be done as  
Runnable r2 = ()->{  
    System.out.println("Hello");  
};  
// can be simplified as  
Runnable r2 = ()->System.out.println("Hello");
```

If the method body has only one statement, then we do not need the curly braces. If the only statement is a return statement, then the return keyword is also not required.

```
Comparator<Book> cmp = new Comparator<>(){  
    public int compare(Book b1, Book b2){  
        return Double.compare(b1.getPrice(), b2.getPrice());  
    }  
}  
// can be simplified as  
Comparator<Book> cmp = (Book b1, Book b2) -> {  
    return Double.compare(b1.getPrice(), b2.getPrice());  
};
```

```
// can further be simplified as
Comparator<Book> cmp = (b1, b2) -> {
    return Double.compare(b1.getPrice(), b2.getPrice());
};

// can further be simplified as
Comparator<Book> cmp = (b1, b2) -> Double.compare(b1.getPrice(),
b2.getPrice());
```

Key/value based data structures in Java

- Unlike a Collection, which is just a bunch of objects, Map is a bunch of objects stored with a unique key
- Key in a Map cannot be duplicate
 - Depending on the implementation of Map, the data type of Key must either
 - implement Comparable (TreeMap)
 - provide hashCode and equals methods (HashMap, Hashtable, LinkedHashMap)
- Map can produce different types of Collection
 1. Set of keys
 2. Collection of values
 3. Set of Entry (key/value pair)

```
@startuml

interface Map<K, V> {
    + put(key: K, value: V): V
    + get(key: K): V
    + isEmpty(): boolean
    + size(): int
    + containsKey(key: K): boolean
    + containsValue(value: V): boolean
    + remove(key: K): V
    + clear(): void
    + keySet(): Set<K>
    + values(): Collection<V>
    + entrySet(): Set<Entry<K, V>>
}

abstract class AbstractMap{}
abstract class Dictionary {}

Map <|-- AbstractMap
Map <|-- HashMap
Map <|-- Hashtable
Map <|-- LinkedHashMap
AbstractMap <|-- HashMap
HashMap <|-- LinkedHashMap
Dictionary <|-- Hashtable
```

```
Hashtable <|-- Properties

interface SortedMap{}
interface NavigableMap{}

Map <|-- SortedMap
SortedMap <|-- NavigableMap

AbstractMap <|-- TreeMap
NavigableMap <|.. TreeMap

@enduml
```

Different implementations of Map

1. HashMap

- Uses an array called buckets
- A bucket contains the value corresponding to a key, and the index of the bucket is determined by the key
 - so, if two keys are identical then they correspond to the same index of the bucket,
 - for example,

```
m.put("vinod", "bangalore");
m.put("vinod", "shivamogga");
```

Here "vinod" as a key would generate an index where initially, "bangalore" is stored. When the second **put** is called, on the index, "shivamogga" will be placed, overwriting "bangalore".

- Refer https://en.wikipedia.org/wiki/Hash_table for more details

2. LinkedHashMap

- Uses a linked list
- uses the hashCode and equals of keys to check for duplicates

3. TreeMap

- uses red-black tree
- depends on Comparable to compare two keys

4. Hashtable

- legacy (version 1.0)
- HashMap is a derivative of this
- some methods are synchronized, and hence this is preferred in a multithreaded applications

5. Properties

- subtype of Hashtable
- key/value pair of strings

