```
interface Map<Key, Value> {









}
```

```
int hash1(String s) {
  return s.length();
}



int hash2(String s) {
  int hash = 0;
  for(int i = 0; i < s.length(); i += 1) {
    hash += Character.codePointAt(s, i);
  }
  return hash;
}



public int hash3(String s) {
  int h = 0;
  for (int i = 0; i < s.length(); i++) {
    h = 31 * h + Character.codePointAt(s, i);
  }
  return h;
}
```

hashCode

`public int hashCode()`

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by `HashMap`.

The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the `equals(java.lang.Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

As much as is reasonably practical, the hashCode method defined by class `Object` does return distinct integers for distinct objects. (The hashCode may or may not be implemented as some function of an object's memory address at some point in time.)

Returns:

a hash code value for this object.

See Also:

`equals(java.lang.Object)` , `System.identityHashCode(java.lang.Object)`