BCS 345 Java Programming

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

- toString
- equals
- hashCode

Today's Lecture

toString

- First we will cover the toString method…

Object toString Implementation

Object.toString() Method

toString() – Returns a string representation of the object.

- There is a default implementation of toString() defined on the Object class.
- **The default implementation will return a string that contains the object type concatenated with an integer.**

Object

```
String toString() {
        Returns string containing class name
        and integer
}
```

Object toString Implementation

Using Default toString Implementation

```
public class Employee {
        public String firstName;
        public String lastName;
}

public class Driver {
        public static void main(String[] args) {
                Employee e1 = new Employee();
                e1.firstName = "Arthur";
                e1.lastName = "Hoskey";
                System.out.println(e1.toString());
        }
}
```

Prints the class name and a number

Prints something like:

bcs345.hoskey.compare.tostring.Employee@4fee225

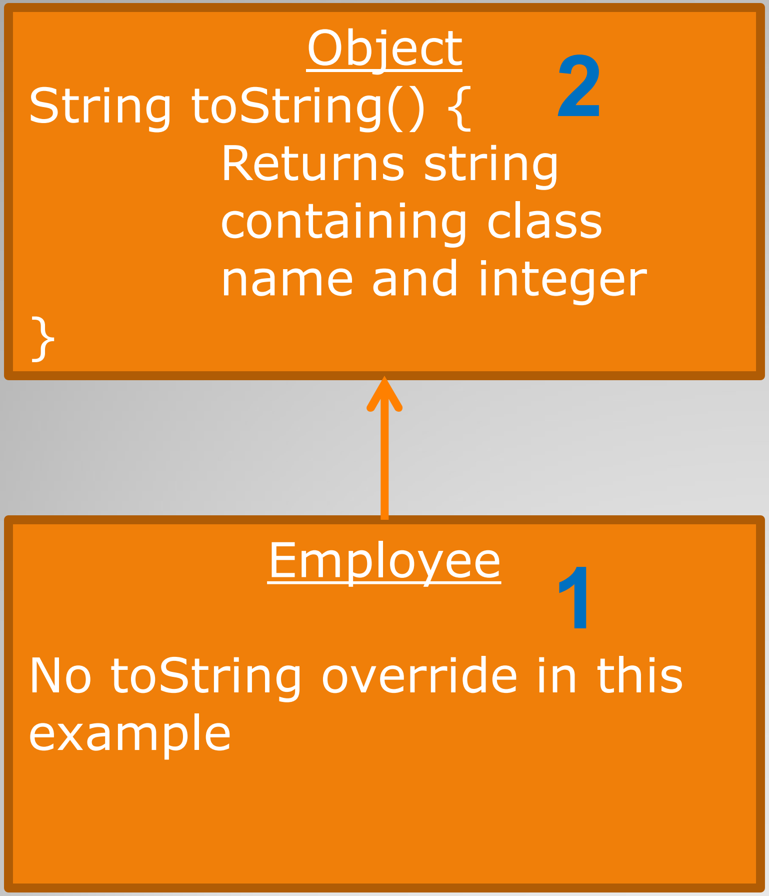
```
public class Employee {
            public String firstName;
            public String lastName;
}

public class Driver {
            public static void main(String[] args) {
                        Employee e1 = new Employee();
                        e1.firstName = "Arthur";
                        e1.lastName = "Hoskey";
                        System.out.println(e1);
            }
}
```

Calls toString automatically!!!

Prints something like:

bcs345.hoskey.compare.tostring.Employee@4fee225

Passing a reference type to print will automatically call toString


Object
String toString() {
Returns string containing class name and integer
}
2
Employee
1
No toString override in this example

toString Method Resolution Example

1. Compiler checks Employee for a toString method implementation (does not exist).
2. Compiler checks base class (Object) for a toString method implementation (finds it)

Compiler will use the Object toString method implementation.

toString Method Resolution

Override toString

Overrides toString in Employee

```
public class Employee {
        public String firstName;
        public String lastName;


        @Override
        public String toString()  {
                String s = firstName + " " + lastName;
                return s;
        }
}
```

Use @Override annotation when overidding

- toString will return a string that contains the first and last names separated by a space.

```
public class Driver {
        public static void main(String[] args) {
                Employee e1 = new Employee();
                e1.firstName = "Arthur";
                e1.lastName = "Hoskey";
                System.out.println(e1.toString());
        }
}
```

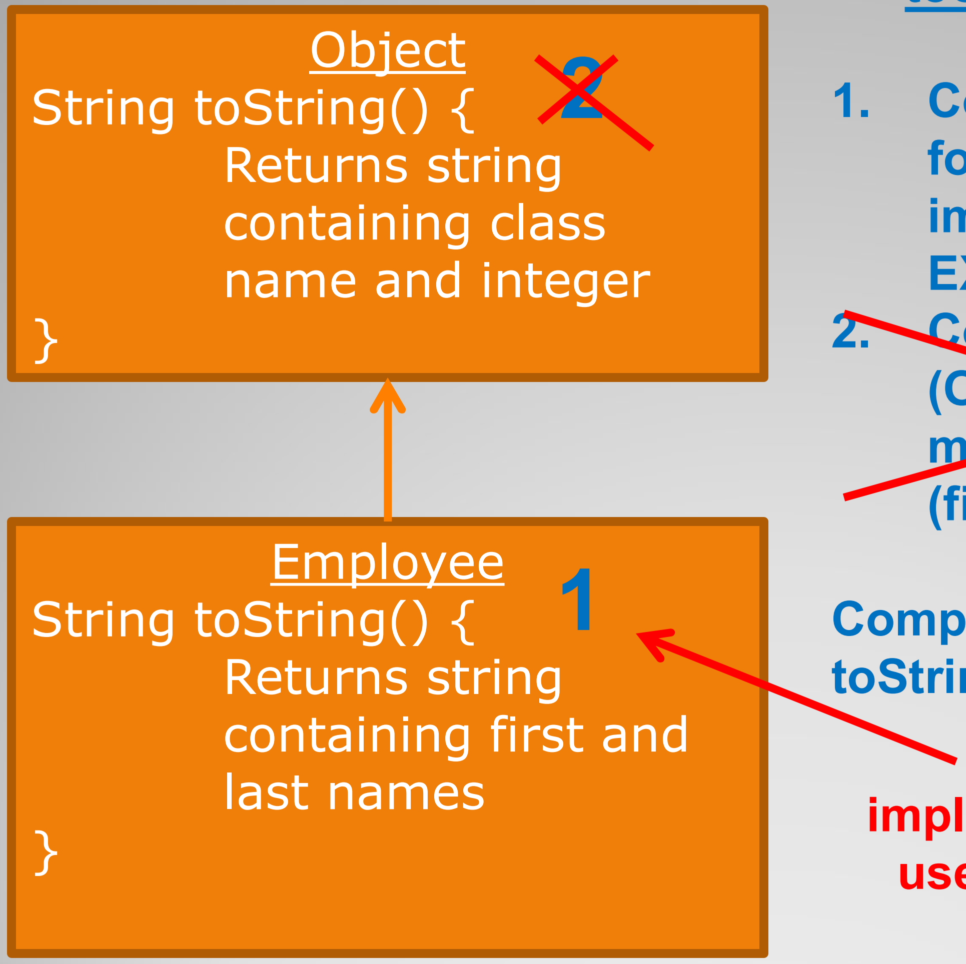
Prints:

```
Arthur Hoskey
```

Uses Employee override of toString (we added it on previous slide)

Using toString Override

toString Method Resolution



```
Object
String toString() {
        Returns string
        containing class
        name and integer
}

Employee
String toString() {
        Returns string
        containing first and
        last names
}
```

toString Method Resolution Example

1. **Compiler checks Employee for a toString method implementation (DOES EXIST!!!).**
2. ~~**Compiler checks base class (Object) for a toString method implementation (finds it)**~~

Compiler will use the Employee toString method implementation.

Finds an Employee implementation of toString so it uses it. No need to check the base class.

Created by Arthur Hoskey, PhD

- Now we will cover the equals method…

equals

```
public class Driver {
        public static void main(String[] args) {
                Employee e1 = new Employee();
                Employee e2 = new Employee();
                e1.firstName = "Arthur";
                e2.firstName = "Arthur";
                e1.lastName = "Hoskey";
                e2.lastName = "Hoskey";
                if (e1 == e2) {
                        System.out.println("Equal");
                } else {
                        System.out.println("Not equal");
                }
        }
}
```

$$\xleftarrow{\text{Compares addresses}}$$

Prints:

Not equal

Object Compare (==)

Object equals Implementation

<u>Object.equals() Method</u>

equals() – Indicates whether some other object is "equal to" this one.

- There is a default implementation of equals() defined on the Object class.
- **The default implementation will test if the addresses of the objects are equal.**

```
public class Driver {
	public static void main(String[] args) {
		Employee e1 = new Employee();
		Employee e2 = new Employee();
		e1.firstName = "Arthur";
		e2.firstName = "Arthur";
		e1.lastName = "Hoskey";
		e2.lastName = "Hoskey";
		if (e1.equals(e2)) {
			System.out.println("Equal");
		} else {
			System.out.println("Not equal");
		}
	}
}
```

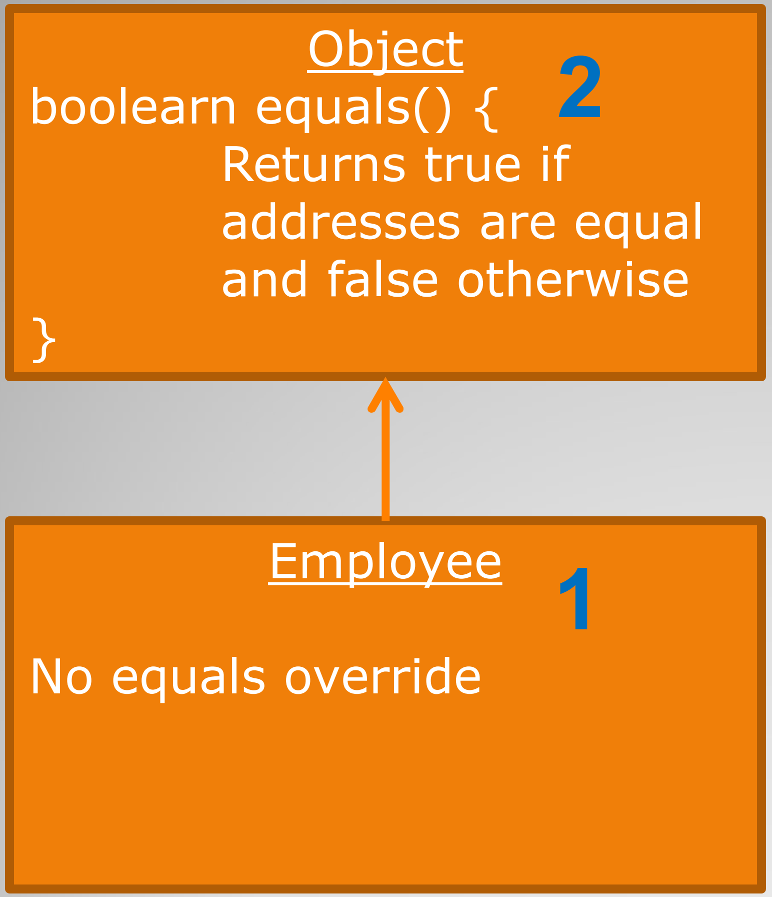
No Employee equals method so it calls Object equals (address compare is used)

Prints:

Not equal

Note: The call to equals is being done on an Employee object (NOT A STRING OBJECT)

Object Compare (default equals)


Object 2

```
boolearn equals() {
        Returns true if
        addresses are equal
        and false otherwise
}
```

Employee 1

No equals override

equals Method Resolution Example

1. Compiler checks Employee for an equals method implementation (does not exist).
2. Compiler checks base class (Object) for an equals method implementation (finds it)

Compiler will use the Object equals method implementation.

Equals Method Resolution

```
public class Employee {
        public String firstName;
        public String lastName;

        @Override
        public boolean equals(Object obj) {
                Employee other = (Employee) obj; // Copy to Employee var

                if (firstName.equals(other.firstName) == false)
                        return false;

                if (lastName.equals(other.lastName) == false)
                        return false;

                return true;
        }
}
```

Override equals in Employee

firstName and lastname are Strings so it is calling the String.equals method which performs a string value comparison

- equals will return true if both firstname and lastname string values are the same and false otherwise.

Override equals

```
public class Driver {
        public static void main(String[] args) {
                Employee e1 = new Employee();
                Employee e2 = new Employee();
                e1.firstName = "Arthur";
                e2.firstName = "Arthur";
                e1.lastName = "Hoskey";
                e2.lastName = "Hoskey";
                if (e1.equals(e2)) {
                        System.out.println("Equal");
                } else {
                        System.out.println("Not equal");
                }
        }
}
```

We added equals implementation to Employee so that one will be used!!!

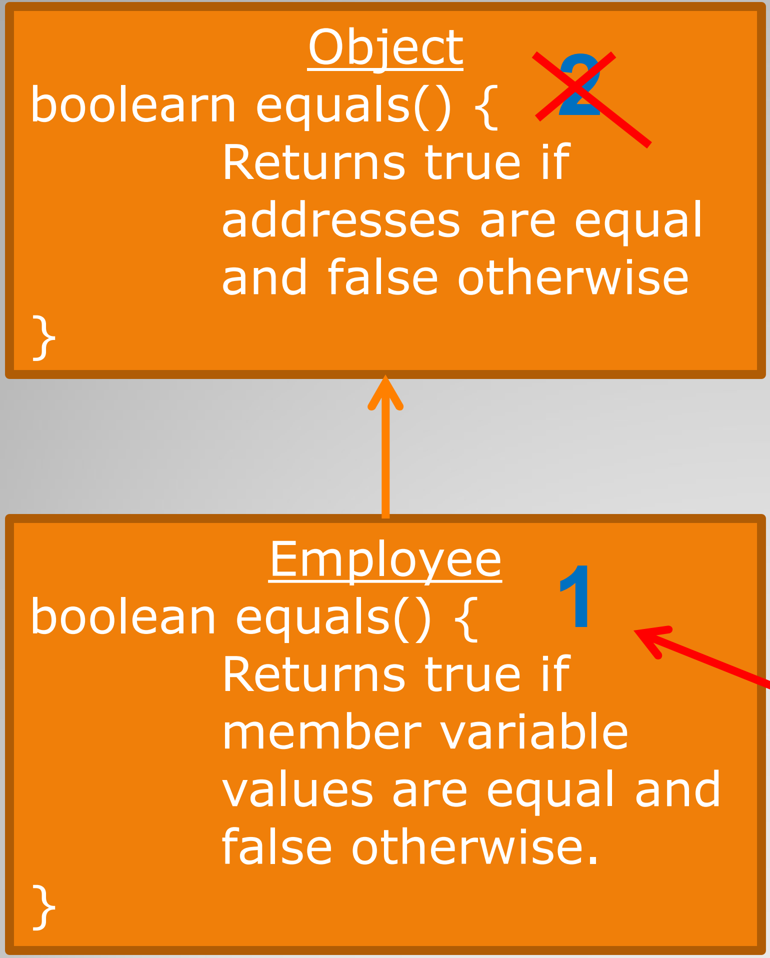
Prints:

```
Equal
```

Note: The call to equals is being done on an Employee object (NOT A STRING OBJECT)

Value Compare

equals Method Resolution



Object
```
boolearn equals() {
        Returns true if
        addresses are equal
        and false otherwise
}
```

Employee
```
boolean equals() {
        Returns true if
        member variable
        values are equal and
        false otherwise.
}
```

equals Method Resolution Example

1. **Compiler checks Employee for an equals method implementation (DOES EXIST!!!).**
2. ~~**Compiler checks base class (Object) for an equals method implementation (finds it)**~~

Compiler will use the Employee equals method implementation.

Finds an Employee implementation of equals so it uses it. No need to check the base class.

String.equals() Method

equals() – The String class provides an override of the equals method. This override compares string values (not addresses).

- This is why the equals method works on strings.
- **For equals to work on classes that you create you must override it yourself. You will have to add code to compare the member variable values or whatever else you want to test.**

String equals

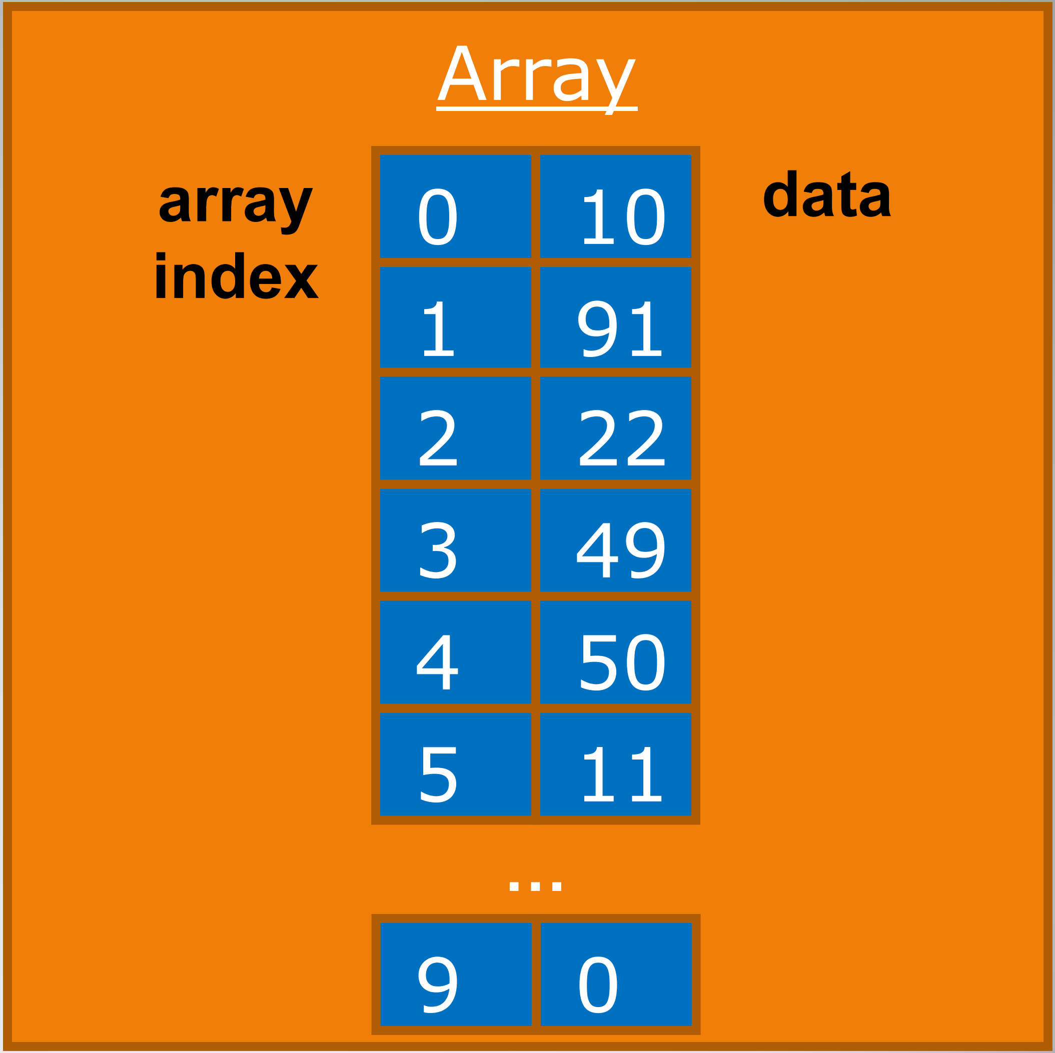
Hash Table

- Now we will cover hash tables in general (prerequisite for describing Java hashCode method)…

Array (Find)

- How would you go about checking the array for the value 49?
- (Advanced) What is the big O runtime of this operation?

Note: If you have not taken a data structures course do not worry about the big O runtime.



Array (Find)

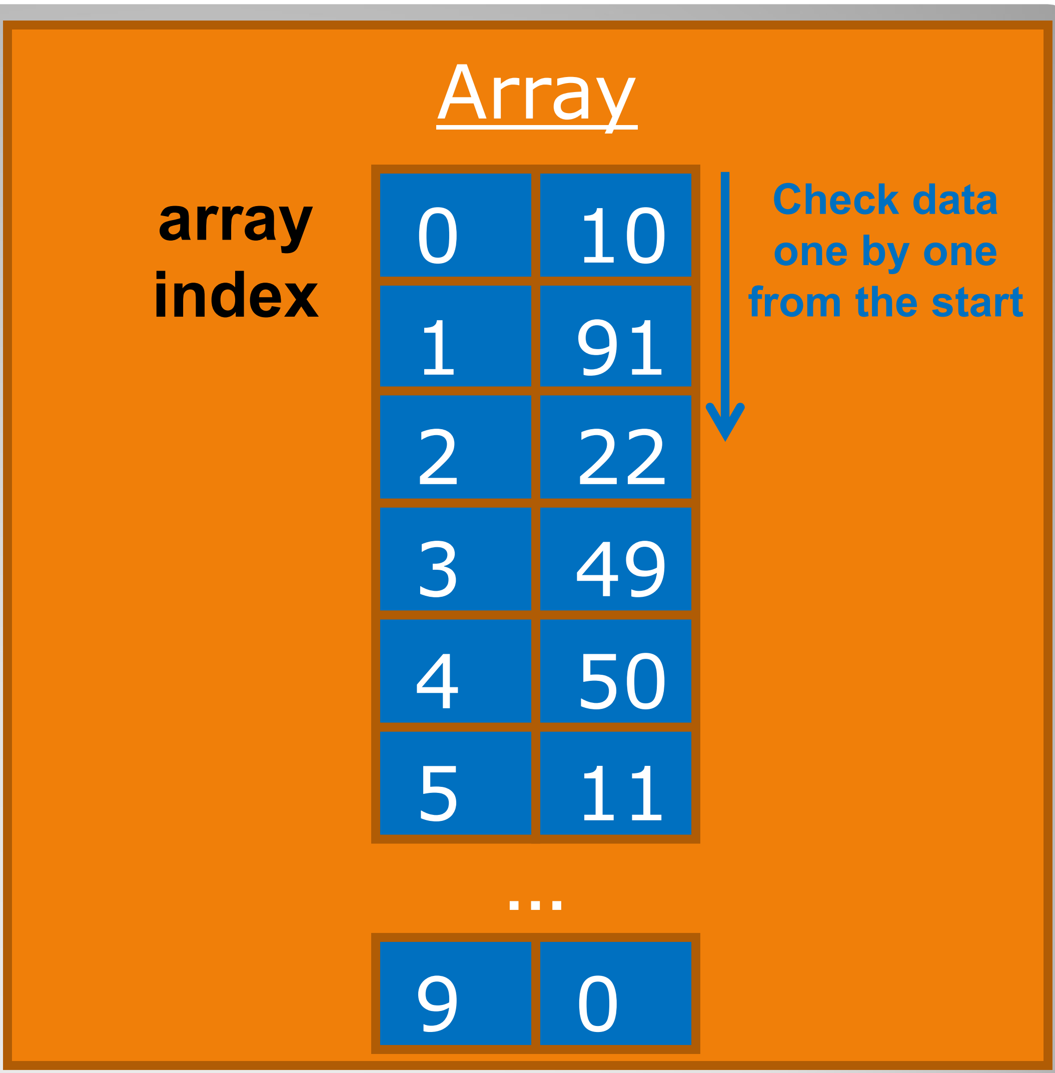
- How would you go about checking the array for the value 49?

Answer: Start from the beginning of the array and check each element to see if it has 49 in it.

- (Advanced) What is the big O runtime of this operation?

Answer: O(n)

We must search the whole array in the worst case

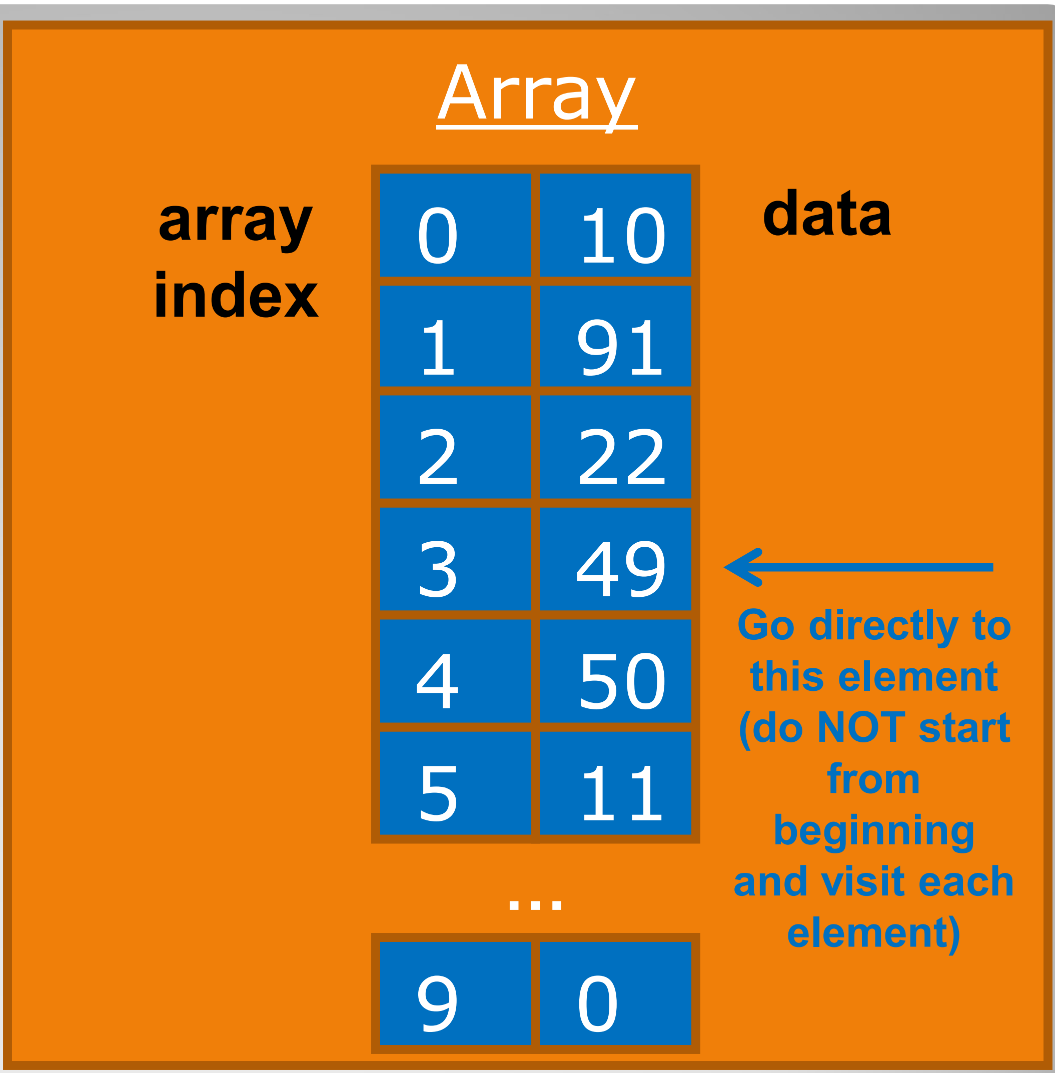


- Now assume we know the index of where the element we are searching for is located.
- If we did know the index then we could just go directly to that element.
- For example…

What if we knew the target element's index?

Array (Find)

- Assume that we know 49 is at index 3.
- We can access it directly
- We do not need to "visit" indexes 0, 1, or 2 to get there (arrays have random access).
- (Advanced) The runtime of this type of access is O(1).


Array
array index
data
0 10
1 91
2 22
3 49
4 50
5 11
...
9 0
Go directly to this element (do NOT start from beginning and visit each element)

Hash Table

<u>Hash Table</u>

- Hash tables provide a mechanism to allow direct access to a piece data (no searching from the beginning of an array).
- A hash table uses a function to convert the data we are searching for into an index. This function is called a **hash function**. This is the "magic" we use to go directly to an index.

- Unfortunately, multiple pieces of data could end up having the same index be returned by the hash function. This is called a **collision**.
- To remedy the collision problem each index is treated as a bucket that can contain multiple values.
- Once a bucket is found, the bucket must then be searched for the target value (this is not a big deal though).

- For example…

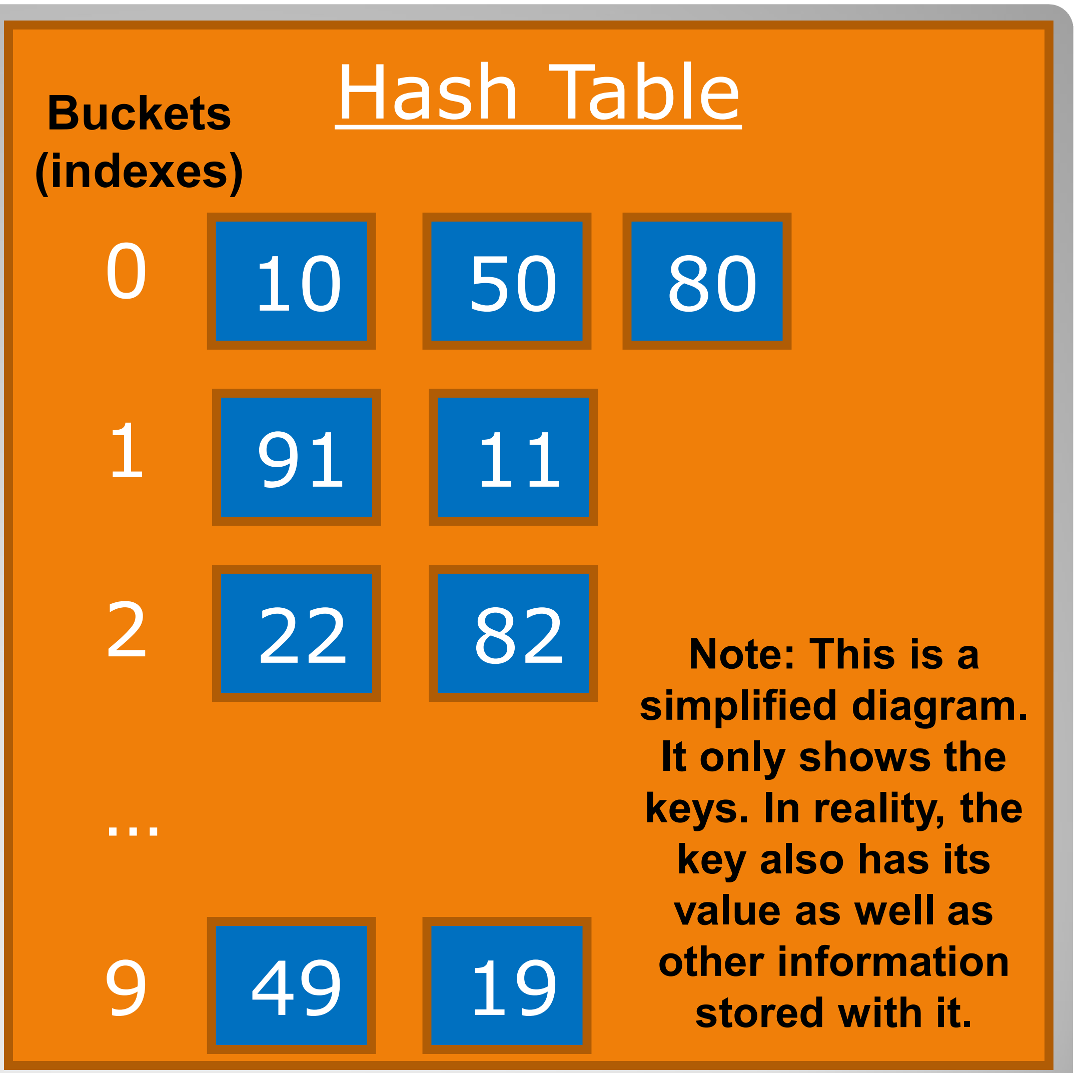
Hash Table

- A hash function is used to determine which bucket to put data into (the data is a key).
- Keys added to this table are:

10, 91, 22, 49, 50, 11, 82, 19, 80

- The hash function in this example is the mod function.

Bucket = Key % NumBuckets



Note: This is a simplified diagram. It only shows the keys. In reality, the key also has its value as well as other information stored with it.

Hash Table – Add

- Hash function:

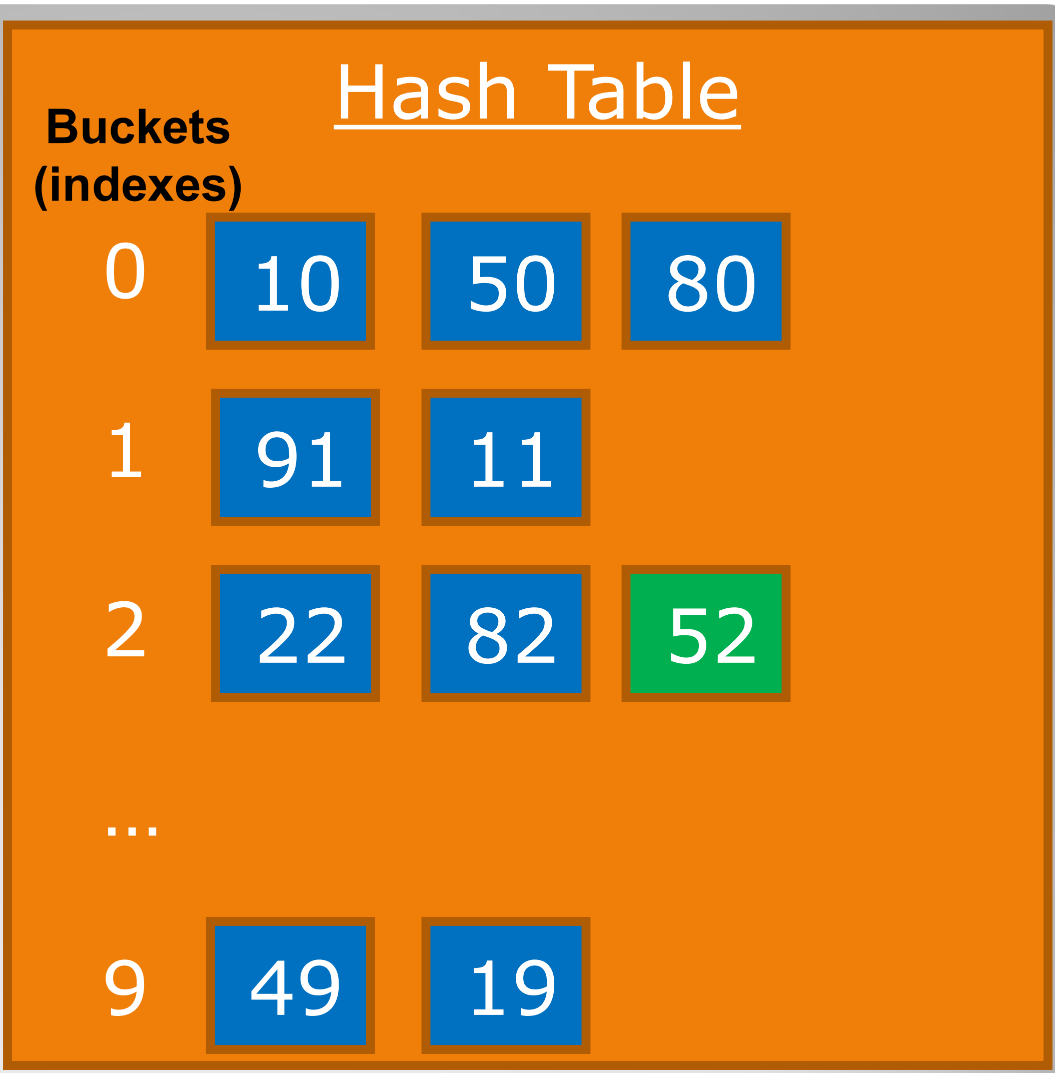
Bucket = Key % NumBuckets

- Now add the key 52 to the hash table:

Key = 52
NumBuckets = 10

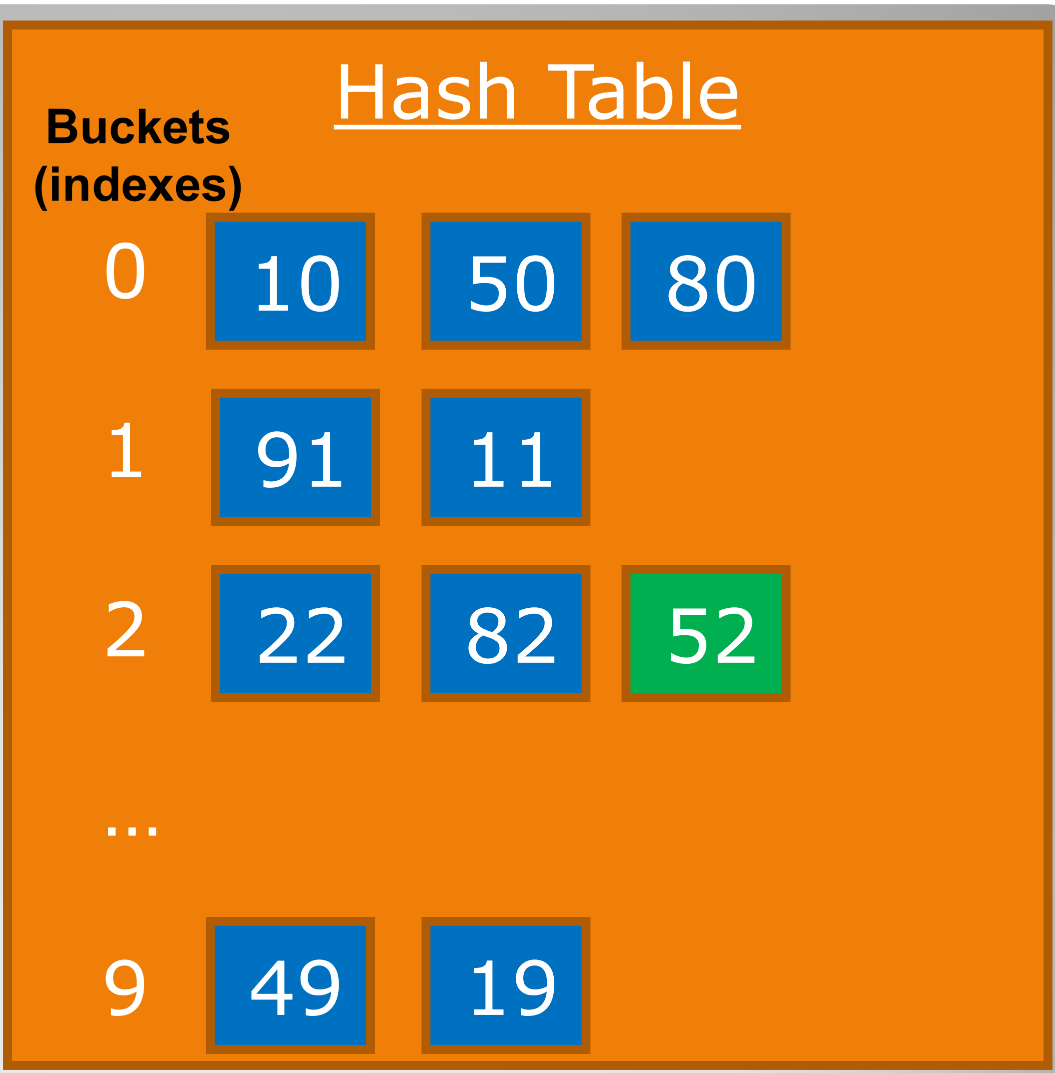
Bucket = Value % NumBuckets
2 = 52 % 10

- Key 52 → Bucket 2



Hash Table – Add

- Add 71???



Hash Table – Add

- Hash function:

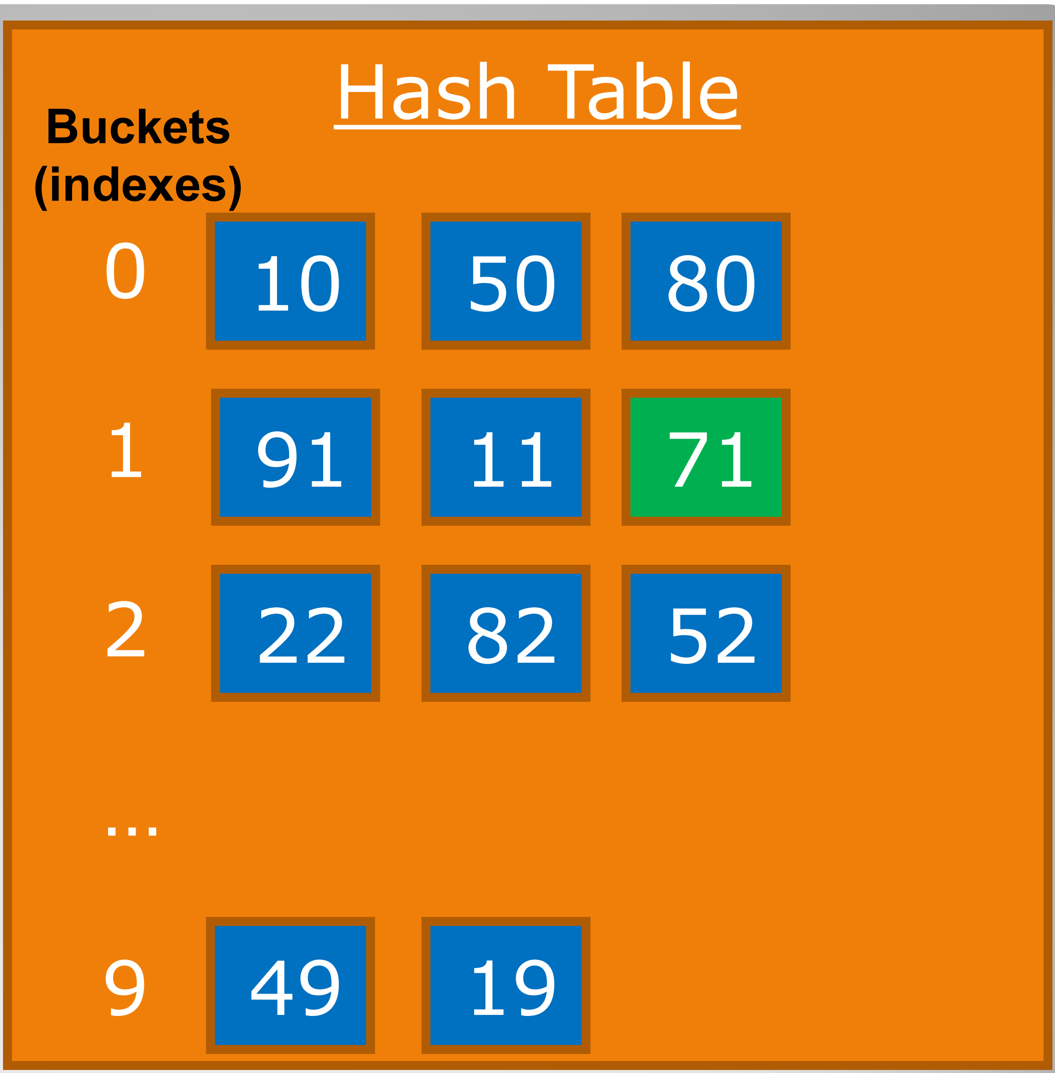
Bucket = Key % NumBuckets

- Now add the key 71 to the hash table:

Key = 71
NumBuckets = 10

Bucket = Value % NumBuckets
1 = 71 % 10

- Key 71 → Bucket 1



Hash Table – Search

- Hash function:

Bucket = Key % NumBuckets

- Now do a search for 49.

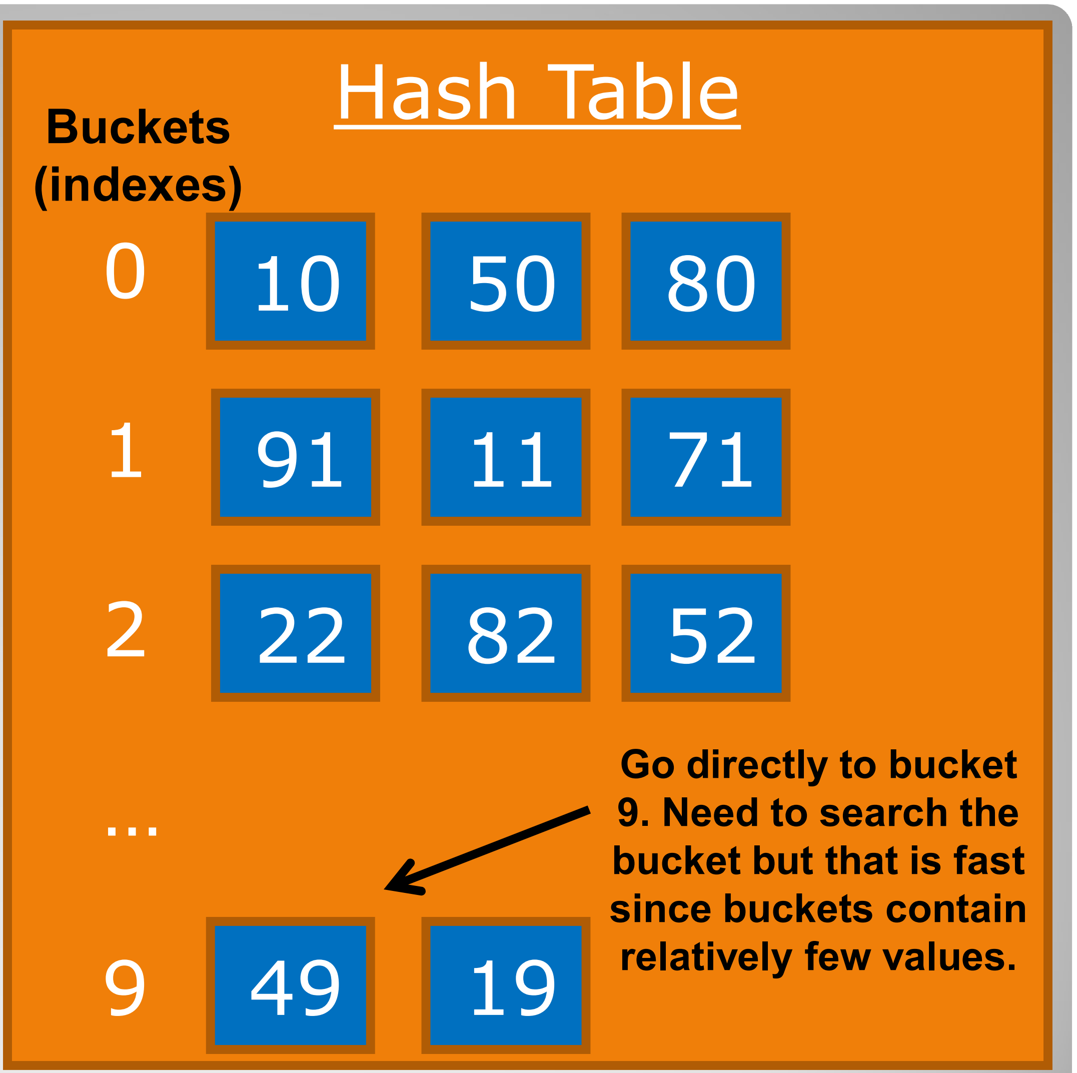
Key = 49

NumBuckets = 10

Bucket = Value % NumBuckets

9 = 49 % 10

- Key 49 → Bucket 9


Hash Table
Buckets (indexes)
0
10
50
80
1
91
11
71
2
22
82
52
…
9
49
19
Go directly to bucket 9. Need to search the bucket but that is fast since buckets contain relatively few values.

Hash Table Search Speed

- Hash tables are primarily used for their fast search speed.
- O(1) search time on average (assuming a hash function that uniformly distributes elements in buckets).

Hash Table Search Speed

- Now we will cover the Java hashCode method…

Java hashCode

Java hashCode

- Java provides a hashCode method that serves as a hash function.
- The hashCode method is used by Java's hash-based collections.
- IMPORTANT! - If you provide an override for equals in a class you should also provide an override for hashCode.

Object.hashCode() Method

- The Object.hashCode method returns a hash value for an object.

hashCode() – Returns a hash code for the current instance.

- **Object's default implementation of hashCode generally returns the address of the object instance.**

Object hashCode Implementation

hashCode Properties

- If two object's return the same value from equals they MUST also return the same value from hashCode.

Same equals value → Same hashCode value

- In contrast, if two objects return the same value from hashCode they do not necessarily return the same value from equals. This is possible because objects with different values can have the same hashCode value (they would go in the same bucket).

~~Same hashCode value → Same equals value~~

hashCode Properties

```
public class Employee {
    public String firstName;
    public String lastName;

    @Override
    public int hashCode() {
        return firstName.hashCode() * lastName.hashCode();
    }

    @Override
    public boolean equals(Object obj) {
        Employee other = (Employee) obj; // Copy to Employee var

        if (firstName.equals(other.firstName) == false)
                return false;

        if (lastName.equals(other.lastName) == false)
                return false;

        return true;
    }
}
```

hashCode Implementation

hashCode using a constant

- What effect does the following hashCode implementation have on a hash table?

```
@Override
public int hashCode() {
  return 1;
}
```

hashCode using a constant

hashCode using a constant

hashCode using a constant

- We could write a hashCode method that just returns a constant. For example:

```
@Override
public int hashCode() {
   return 1;
}
```

$\xleftarrow{\quad}$ **Always returns 1. This is bad!!! Will NOT evenly distribute elements in buckets.**

- This would cause all elements to be placed in the same bucket in the hash table (hash table deteriorates into a list).
- Since it is effectively a list the search time is now O(n).
- This defeats the purpose of using the hash table in the first place (hash tables are used for their fast searches).
- **Moral of the Story - The hashCode function must evenly distribute elements in buckets to ensure a fast search.**

Hash Table vs Hash Sets

- **Hash tables are designed around calculating an index so we can go directly to the data.**

Hash Table (general description)

- Maps keys to values (associative array). Will store key/value pairs.

Hash Set (general description)

- An unordered collection of values (not key/value pairs).

- Both are good for searching for values
- Both use "buckets" to store data (fixes collisions)
- How it works:

Given a key the hash function transforms the key into a bucket number then it stores/searches in ONLY that bucket.

- End of Slides

End of Slides