

# CSE 12 – Basic Data Structures and Object-Oriented Design

## Lecture 19

Greg Miranda, Spring 2021

# Announcements

- Quiz 19 due Friday @ 12pm
- PA6 due tonight @ 11:59pm (open)
- Survey 7 due Friday @ 11:59pm
- Exam 2 – Week 8
  - Released Friday 5/21 @ 2pm
  - Due Saturday 5/22 @ 6pm
  - Topics:
    - Cumulative
    - Big topics – lectures 9 - 17
      - Big O, Big Theta run-time analysis
      - Sorting algorithms
      - Hash tables/maps

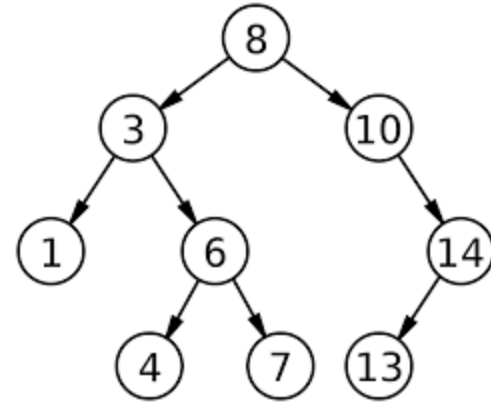
# Topics

- Binary Search Trees
- Questions on Lecture 19?

# Binary Search Tree

What order does PAE() traverse the tree?

```
void printAllElements(Node<K, N> n) {  
    if (n == null ) return;  
    System.out.println(n.key);  
    printAllElements(n.left);  
    printAllElements(n.right);  
}  
  
void printAllElement() {  
    printAllElements(this.root);  
}
```



What's the post, pre, in-order traversal of this tree?

```

class Node<K,V> {
    K key;
    V value;
    Node<K,V> left;
    Node<K,V> right;
    public Node(K key, V value,
                Node<K,V> left,
                Node<K,V> right) {
        this.key = key;
        this.value = value;
        this.left = left;
        this.right = right;
    }
}

```

```

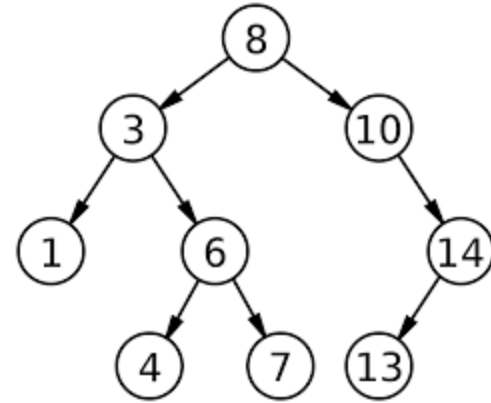
class BSTMap<K,V> implements OrderedDefaultMap<K,V>{
    Node<K, V> root;
    int size;
    Comparator<K> comparator;
    ...
    Node<K, V> set(Node<K, V> node, K key, V value) {
        if (node == null) {
            this.size += 1;
            return new Node<K, V>(key, value, null, null);
        }
        int comp = this.comparator.compare(node.key, key);
        if (comp < 0) {
            node.right = this.set(node.right, key, value);
            return node;
        } else if (comp > 0) {
            node.left = this.set(node.left, key, value);
            return node;
        } else {
            node.value = value;
            return node;
        }
    }
}

@Override
public void set(K key, V value) {
    if (key == null) {
        throw new IllegalArgumentException();
    }
    this.root = this.set(this.root, key, value);
}
}

```

# Binary Search Tree

- Assume the key and value are identical for this example
- `set("5", 5);`
- `set("11", 11);`
- `set("15", 15);`
- `set("12", 12);`
- What's the picture after calling the above `set()` methods?



# Wildcards

## Hope

- Our generic class should take any type that is a subtype of `E`
- And we hope that `findFirst` can take `ArrayList` of any subtype of `E`

## But

- Current generic system doesn't allow that.

```
public class WildCardsExe <E extends Person>{  
    public E findFirst(ArrayList<E> list)
```

Java provides a flexible type – the wildcard – ?

`<?>` means any type

`Collection<?>` means Collection of any type

```
public class WildCardsExe <E extends Person>{

    public E findFirst(ArrayList<? extends E> list){
        if (list == null || list.size() == 0){
            return null;
        }
        return list.get(0);
    }

    public static void main(String[] args){
        WildCardsExe<Person> ref = new WildCardsExe<Person>();
        ArrayList<Person> pList = new ArrayList<Person>();
        pList.add(new Person("PC"));
        pList.add(new Person("HA"));
        System.out.println(ref.findFirst(pList));
        ArrayList<Student> sList = new ArrayList<Student>();
        sList.add(new Student("PC", 11));
        sList.add(new Student("HA", 33));
        System.out.println(ref.findFirst(sList));
    }
}
```

**? : unbounded wildcard** represents any subtype of E so our ArrayList is more general (it implies ? extends Object)

**? extends E : bounded wildcard** represents E or any subtype of E

**? super E: lower-bounded wildcard** represents E or any super type of E



```
void doIt(Collection<? extends Student> data){  
    for (Student s: data){  
        System.out.println(s)  
    }  
}
```

**Does the following code compile?**

```
Collection<Student> data = new ArrayList<Student>();  
doIt(data);
```

- A. Yes
- B. No

**Does the following code compile?**

```
Collection<Person> data = new ArrayList<Person>();  
doIt(data);
```

```
void doIt(Collection<? extends Student> data){  
    for (Student s: data){  
        System.out.println(s)  
    }  
}
```

**Does the following code compile?**

```
Collection<Student> data = new ArrayList<Student>();  
doIt(data);
```

**Does the following code compile?**

```
Collection<Person> data = new ArrayList<Person>();  
doIt(data);
```

**How do we change doIt such that it will work for both situations**

- A. change parameter to `Collection<? extends Person> data`
- B. change parameter to `Collection<? super Student> data`
- C. change parameter to `Collection<? super Person> data`
- D. change foreach loop to `for (Object s: data)`
- E. Some combination of the above

# Unbounded wildcard – ‘?’

```
static void soundOff(Collection<?> listOfAnimals) {  
  
    for (Animal a : listOfAnimals) {  
  
        a.makeNoise();  
  
    }  
  
}
```

```
Collection<Dog> dogList = new ArrayList<Dog>();  
soundOff(dogList);
```

Does this solve our problem?

- A. Yes, this code will work
- B. No, this code has a compile error

addAll should accept collections that contain any type that 'is-a' E.

```
public abstract class AbstractCollection<E>
    implements Collection<E> {
    // Add all the elements of the argument Collection
    // to this Collection
    public boolean addAll(_____ c) {
```

- A. Collection<E>
- B. Collection<?>
- C. Collection<? extends E>
- D. Collection<? super E>
- E. More than one of these will work

```

import java.util.*;

public class SuperWildCardDemo {

    public static void main(String[] args) {

        ArrayList<String> list1 = new ArrayList<String>();
        ArrayList<Object> list2 = new ArrayList<Object>();

        list2.add("CSE");

        list2.add(12);

        list1.add("UCSD");

        add(list1, list2);

        System.out.println(list2);

    }

```

What types should I fill into the blanks

- |                   |             |
|-------------------|-------------|
| A. T              | ? extends T |
| B. ? extends T    | T           |
| C. T              | ? super T   |
| D. ? super TT     |             |
| E. Something else |             |

```

public static <T> void add(ArrayList<_____> c1, ArrayList<_____> c2) {

    while (!c1.isEmpty())

        c2.add(c1.remove(0));

}

}

```