INFO1113 / COMP9003
Object-Oriented Programming

Lecture 9

THE UNIVERSITY OF
SYDNEY

Acknowledgement of Country

I would like to acknowledge the Traditional Owners of Australia and recognise their continuing connection to land, water and culture. I am currently on the land of the Gadigal people of the Eora nation and pay my respects to their Elders, past, present and emerging.

I further acknowledge the Traditional Owners of the country on which you are on and pay respects to their Elders, past, present and future.

Copyright Warning

Topics: Part A

- **Recursion**

- **Recursion with OOP**

- **Memoization (Caching Results)**

Recursion

Recursion is a technique within computer science that allows calling a function within itself.

Recursive functions are aligned with recursive sequences or series, where the output of a function is dependent on the output from the same function with a change of input.

Refer to Chapter 11.1 The Basics Of Recursion, p 856-876 (Java, An Introduction to Problem Solving & Programming, Savitch & Mock)

Recursion

Recursive function is made of

- A base case (or many base cases), where the function terminates.

- Recursive case (or many recursive cases), which will converge to a base case.

Recursion

Drawbacks from recursion

- The java programming model does not allow for **infinite** recursion
- Inefficient with memory
- Potentially more computationally demanding due to the overhead caused by method calls

However, problems can often be represented easily with recursion.

Refer to Chapter 11.2 Programming With Recursion, p. 876-890 (Java, An Introduction to Problem Solving & Programming, Savitch & Mock)

Factorial (*N!*)

- *N*! = (*N*-1)! * N   [for *N* > 1]
- 1! = 1
- 3!
    = 2! * 3
    = (1! * 2) * 3
    = 1 * 2 * 3

```
public int factorial(int n)
{
   int fact;
   if (n > 1) // recursive case
      fact = factorial(n - 1) * n;
   else // base case
      fact = 1;

   return fact;
}
```

```
public int factorial(int 3)
{
  int fact;
  if (3 > 1)
    fact = factorial(2) * 3;
  else
    fact = 1;
  return fact;
}
```

```
public int factorial(int 3)
{
   int fact;
   if (3 > 1)
     fact = factorial(2) * 3;
   else
     fact = 1;
   return fact;
}
```

```
public int factorial(int 2)
{
   int fact;
   if (2 > 1)
     fact = factorial(1) * 2;
   else
     fact = 1;
   return fact;
}
```
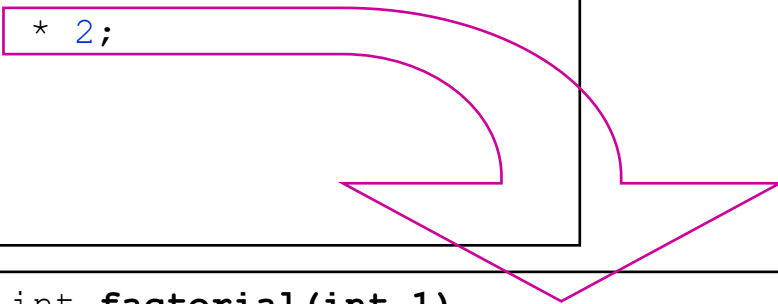
```
public int factorial(int 3)
{
   int fact;
   if (3 > 1)
     fact = factorial(2) * 3;
   else
     fact = 1;
   return fact;
}
```

```
public int factorial(int 2)
{
   int fact;
   if (2 > 1)
     fact = factorial(1) * 2;
   else
     fact = 1;
   return fact;
}
```

```
public int factorial(int 1)
{
   int fact;
   if (1 > 1)
     fact = factorial(n - 1) * n;
   else
     fact = 1;
   return fact;
}
```

```
public int factorial(int 3)
{
   int fact;
   if (3 > 1)
     fact = factorial(2) * 3;
   else
     fact = 1;
   return fact;
}
```

```
public int factorial(int 2)
{
   int fact;
   if (2 > 1)
     fact = factorial(1) * 2;
   else
     fact = 1;
   return fact;
}
```

```
public int factorial(int 1)
{
   int fact;
   if (1 > 1)
     fact = factorial(n - 1) * n;
   else
     fact = 1;
   return 1;
}
```

```
public int factorial(int 3)
{
   int fact;
   if (3 > 1)
     fact = factorial(2) * 3;
   else
     fact = 1;
   return fact;
}
```

```
public int factorial(int 2)
{
   int fact;
   if (2 > 1)
     fact = 1 * 2;
   else
     fact = 1;
   return fact;
}
```
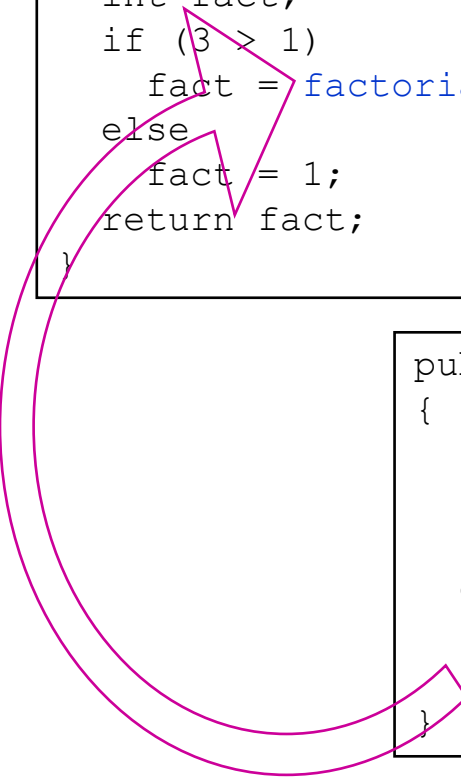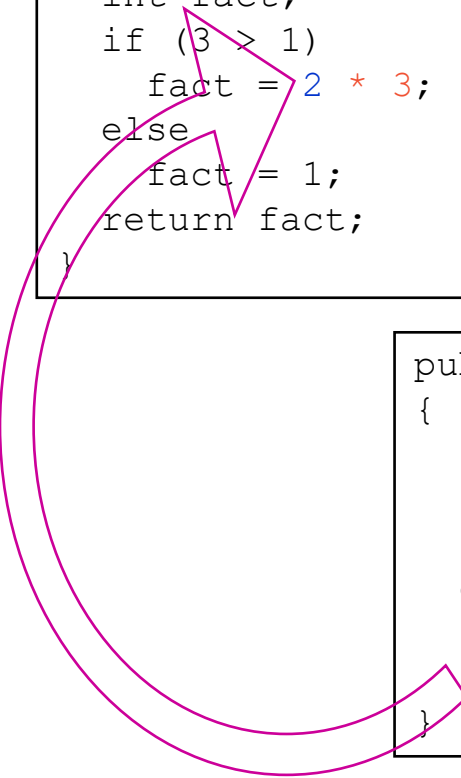
```
public int factorial(int 1)
{
   int fact;
   if (1 > 1)
     fact = factorial(n - 1) * n;
   else
     fact = 1;
   return 1;
}
```

```
public int factorial(int 3)
{
  int fact;
  if (3 > 1)
    fact = factorial(2) * 3;
  else
    fact = 1;
  return fact;
}
```

```
public int factorial(int 2)
{
  int fact;
  if (2 > 1)
    fact = 1 * 2;
  else
    fact = 1;
  return 2;
}
```

```
public int factorial(int 3)
{
   int fact;
   if (3 > 1)
      fact = 2 * 3;
   else
      fact = 1;
   return fact;
}
```
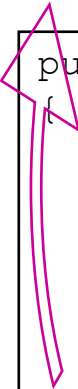
```
public int factorial(int 2)
{
   int fact;
   if (2 > 1)
      fact = 1 * 2;
   else
      fact = 1;
   return 2;
}
```

```java
public int factorial(int 3)
{
  int fact;
  if (3 > 1)
    fact = 2 * 3;
  else
    fact = 1;
  return 6;
}
```

Execution Tree

```
public static int factorial(int n)
{
   int fact;
   if (n > 1) // recursive case
     fact = factorial(n - 1) * n;
   else // base case
     fact = 1;
   return fact;
}
```

```
            factorial(4)

    factorial(3)      4
```

Execution Tree

```
public static int factorial(int n)
{
   int fact;
   if (n > 1) // recursive case
      fact = factorial(n – 1) * n;
   else // base case
      fact = 1;
   return fact;
}
```

factorial(4)

factorial(3)   4

factorial(2)   3

Execution Tree

```
public static int factorial(int n)
{
    int fact;
    if (n > 1) // recursive case
        fact = factorial(n – 1) * n;
    else // base case
        fact = 1;
    return fact;
}
```

factorial(4)

factorial(3)   4

factorial(2)   3

factorial(1)   2
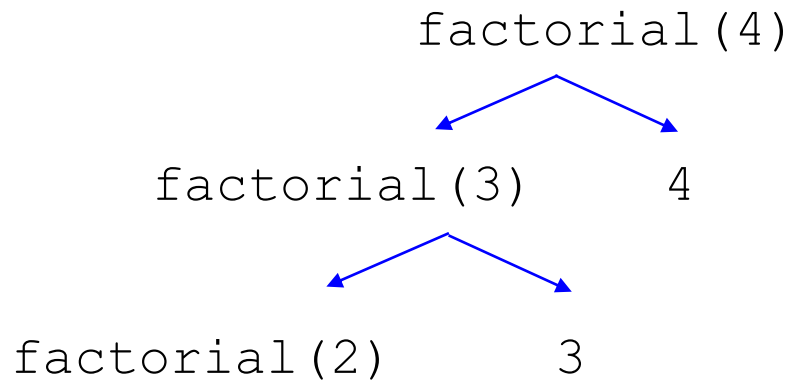
Execution Tree

```
public static int factorial(int n)
{
   int fact;
   if (n > 1) // recursive case
     fact = factorial(n - 1) * n;
   else // base case
     fact = 1;
   return fact;
}
```

```
                    factorial(4)
                         *
        factorial(3)           4
                   *
    factorial(2)         3
              *
factorial(1)->1     2
```
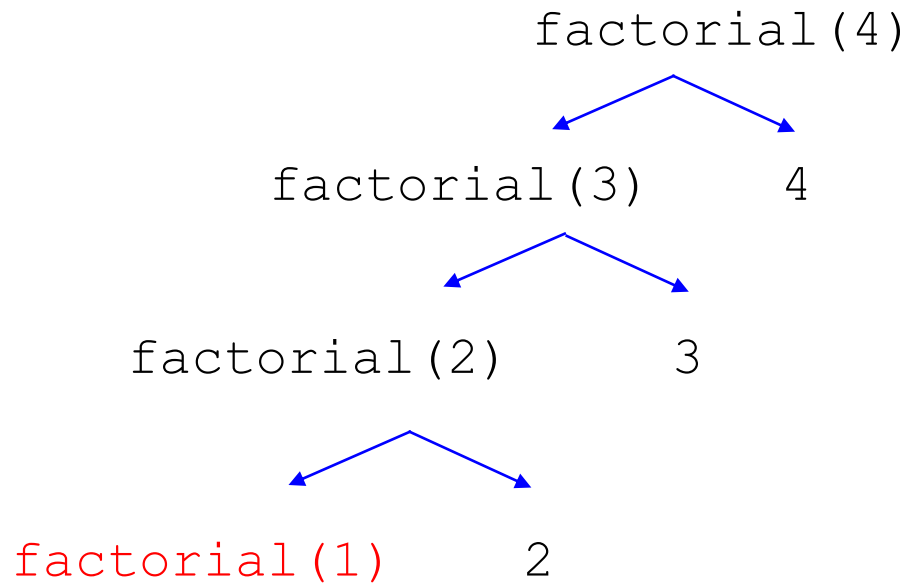
Execution Tree

```
public static int factorial(int n)
{
    int fact;
    if (n > 1) // recursive case
        fact = factorial(n – 1) * n;
    else // base case
        fact = 1;
    return fact;
}
```

```
            factorial(4)
                  *
    factorial(3)        4
              *
factorial(2)->2      3
```

Execution Tree

```
public static int factorial(int n)
{
   int fact;
   if (n > 1) // recursive case
     fact = factorial(n – 1) * n;
   else // base case
     fact = 1;
   return fact;
}
```

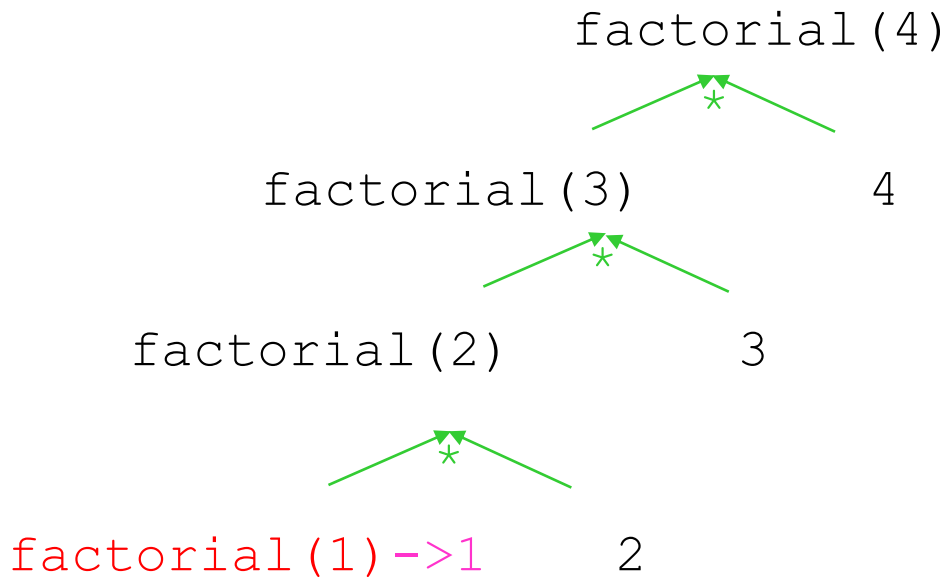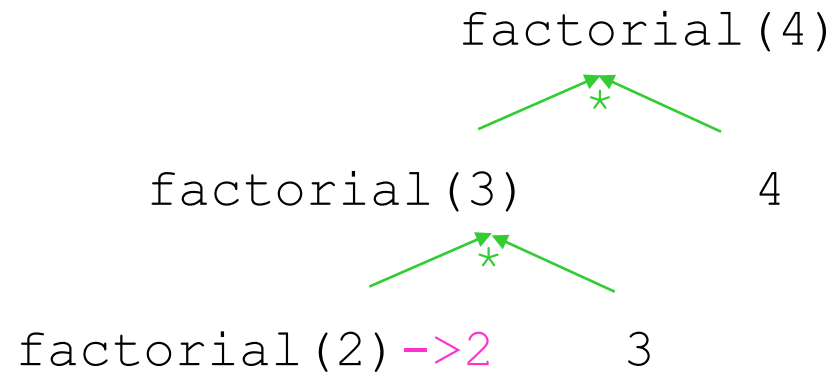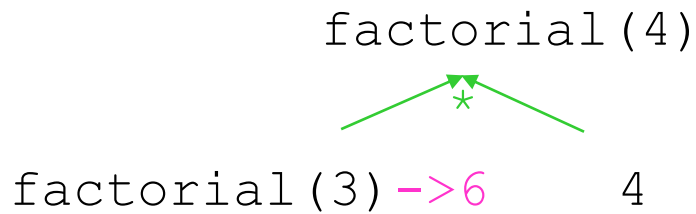factorial(4)

factorial(3)->6   4

Execution Tree

```
public static int factorial(int n)
{
   int fact;
   if (n > 1) // recursive case
     fact = factorial(n - 1) * n;
   else // base case
     fact = 1;
   return fact;
}
```

`factorial(4)->24`

Fibonacci Numbers

- The *N*th Fibonacci number is the sum of the previous two Fibonacci numbers

$$0, 1, 1, 2, 3, 5, 8, 13, \ldots$$

- Recursive case:
    - fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)   for n > 1

  Base case
    - fibonacci(0) = 0
    - fibonacci(1) = 1

```
public int fibonacci(int n)
{
    int fib;
    if (n > 1)
        fib = fibonacci(n-1) + fibonacci(n-2);
    else
        fib = n;
    return fib;
}
```

Execution Tree

```
public int fibonacci(int n)
{
    int fib;
    if (n > 1)
        fib = fibonacci(n-1) + fibonacci(n-2);
    else
        fib = n;
    return fib;
}
```

fibonacci(3)

fibonacci(2)   fibonacci(1)

Execution Tree

```
public int fibonacci(int n)
{
    int fib;
    if (n > 1)
        fib = fibonacci(n-1) + fibonacci(n-2);
    else
        fib = n;
    return fib;
}
```

fibonacci(3)

fibonacci(2)   fibonacci(1)

fibonacci(1)   fibonacci(0)

Execution Tree

```
public int fibonacci(int n)
{
    int fib;
    if (n > 1)
        fib = fibonacci(n-1) + fibonacci(n-2);
    else
        fib = n;
    return fib;
}
```

fibonacci(3)

fibonacci(2)   fibonacci(1)

fibonacci(1)->1   fibonacci(0)->0

Execution Tree

```
public int fibonacci(int n)
{
    int fib;
    if (n > 1)
      fib = fibonacci(n-1) + fibonacci(n-2);
    else
      fib = n;
    return fib;
}
```

fibonacci(3)

+

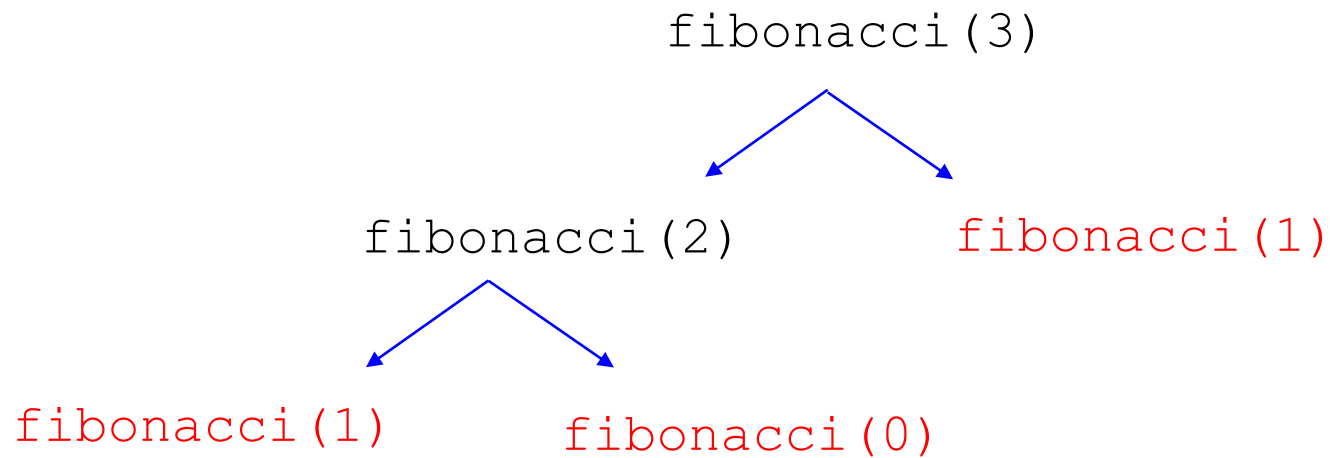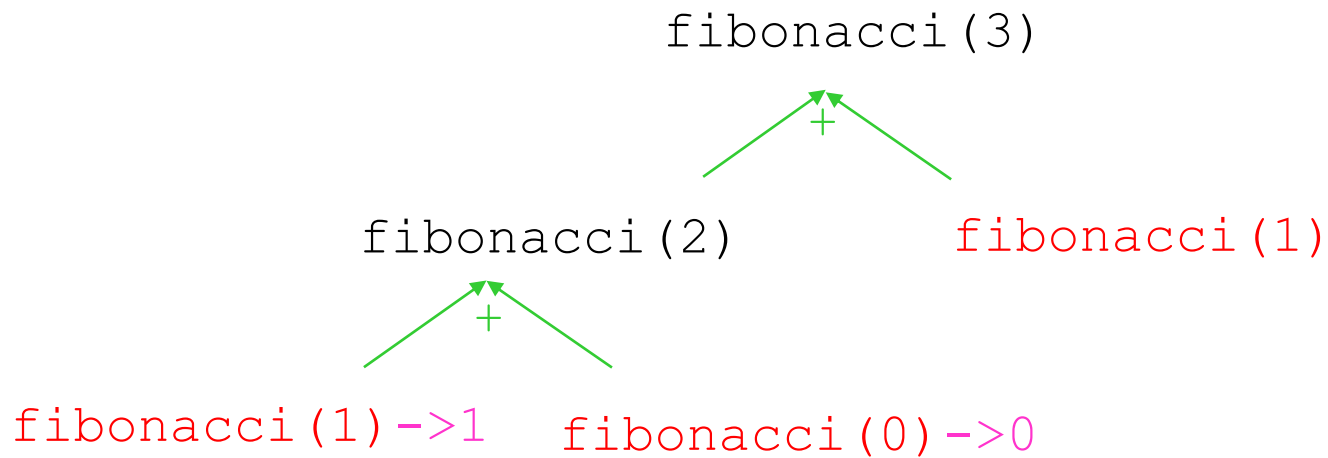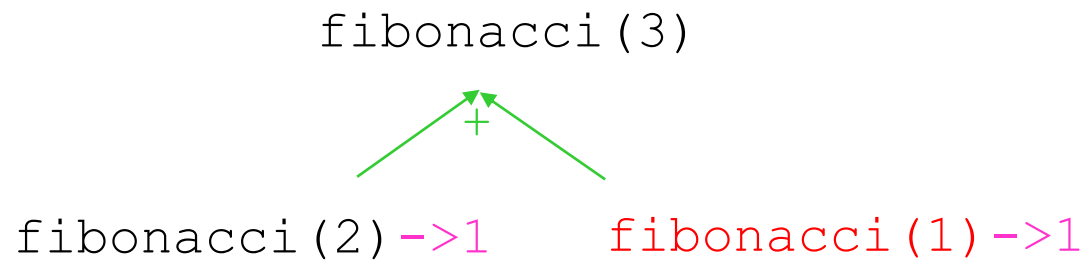fibonacci(2)->1   fibonacci(1)->1

Execution Tree

```
public int fibonacci(int n)
{
    int fib;
    if (n > 1)
        fib = fibonacci(n-1) + fibonacci(n-2);
    else
        fib = n;
    return fib;
}
```

`fibonacci(3)->2`

Fibonacci Sequence

```java
public class Fibonacci {

    public int[] generateSequence(int n) {

        if(n == 0) {
            return new int[] {0};
        } else if(n == 1) {
            return new int[] {0, 1};
        } else {
            int[] s1 = generateSequence(n-1);
            int[] s2 = generateSequence(n-2);
            int[] newS = new int[s1.length+1];
            for(int i = 0; i < s1.length; i++) {
                newS[i] = s1[i];
            }
            newS[newS.length-1] = s1[s1.length-1] + s2[s2.length-1];
            return newS;
        }
    }
}
```

n	0	1	2	3	4	5	6	7	8	9	...
F_n	0	1	1	2	3	5	8	13	21	34	

generateSequence is a recursive method that takes in an integer.

Fibonacci Sequence

n	0	1	2	3	4	5	6	7	8	9	...
F_n	0	1	1	2	3	5	8	13	21	34	

```java
public class Fibonacci {

    public int[] generateSequence(int n) {

        if(n == 0) {
            return new int[] {0};
        } else if(n == 1) {
            return new int[] {0, 1};
        } else {
            int[] s1 = generateSequence(n-1);
            int[] s2 = generateSequence(n-2);
            int[] newS = new int[s1.length+1];
            for(int i = 0; i < s1.length; i++) {
                newS[i] = s1[i];
            }
            newS[newS.length-1] = s1[s1.length-1] + s2[s2.length-1];
            return newS;
        }
    }
}
```

Our base cases
With the base cases we simply return an element to the caller.

Fibonacci Sequence

```java
public class Fibonacci {

    public int[] generateSequence(int n) {

        if(n == 0) {
            return new int[] {0};
        } else if(n == 1) {
            return new int[] {0, 1};
        } else {
            int[] s1 = generateSequence(n-1);
            int[] s2 = generateSequence(n-2);
            int[] newS = new int[s1.length+1];
            for(int i = 0; i < s1.length; i++) {
                newS[i] = s1[i];
            }
            newS[newS.length-1] = s1[s1.length-1] + s2[s2.length-1];
            return newS;
        }
    }
}
```

n	0	1	2	3	4	5	6	7	8	9	...
F_n	0	1	1	2	3	5	8	13	21	34	

In this instance, the recursive method calls the same method with a change of input. **N-1 and N-2**

Our recursive case

Let's demo this!

Recursion with OOP

We are able to write the recursive method for class instances.

This kind of recursion is common with linked data structures such as.

- Trees

- Linked List

- Graphs

- Stacks

- Queues

- Heaps

Recursion with OOP

```java
class FamilyMember {

  private String name;
  List<FamilyMember> children;

  public FamilyMember(String name) {
    this.name = name;
    children = new ArrayList<FamilyMember>();
  }

  public void addChildren(FamilyMember f){
      children.add(f);
  }

  public List<FamilyMember> getAllParents() {
    List<FamilyMember> parents = new ArrayList<FamilyMember>();
    if(children.size() > 0) {
      parents.add(this);
      for(int i = 0; i < children.size(); i++) {
        parents.addAll(children.get(i).getAllParents());
      }
    }
    return parents;
  }
}
```

Recursion with OOP

```java
class FamilyMember {

    private String name;
    List<FamilyMember> children;

    public FamilyMember(String name) {
        this.name = name;
        children = new ArrayList<FamilyMember>();
    }

    public void addChildren(FamilyMember f){
            children.add(f);
    }

    public List<FamilyMember> getAllParents() {
        List<FamilyMember> parents = new ArrayList<FamilyMember>();
        if(children.size() > 0) {
            parents.add(this);
            for(int i = 0; i < children.size(); i++) {
                parents.addAll(children.get(i).getAllParents());
            }
        }
        return parents;
    }
}
```

Normal class with name as per the requirements.

Recursion with OOP

```java
class FamilyMember {

    private String name;
    List<FamilyMember> children;

    public FamilyMember(String name) {
        this.name = name;
        children = new ArrayList<FamilyMember>();
    }

    public void addChildren(FamilyMember f){
        children.add(f);
    }

    public List<FamilyMember> getAllParents() {
        List<FamilyMember> parents = new ArrayList<FamilyMember>();
        if(children.size() > 0) {
            parents.add(this);
            for(int i = 0; i < children.size(); i++) {
                parents.addAll(children.get(i).getAllParents());
            }
        }
        return parents;
    }
}
```

It contains a list of **children** or in a more abstract sense, **links**.

Recursion with OOP

```java
class FamilyMember {

    private String name;
    List<FamilyMember> children;

    public FamilyMember(String name) {
        this.name = name;
        children = new ArrayList<FamilyMember>();
    }

    public void addChildren(FamilyMember f){
        children.add(f);
    }

    public List<FamilyMember> getAllParents() {
        List<FamilyMember> parents = new ArrayList<FamilyMember>();
        if(children.size() > 0) {
            parents.add(this);
            for(int i = 0; i < children.size(); i++) {
                parents.addAll(children.get(i).getAllParents());
            }
        }
        return parents;
    }
}
```

Each FamilyMember contain a list of children, when retrieving a list of parents from a **FamilyMember** we will need to check each link if they are also a parent.

Recursion with OOP

```java
class FamilyMember {

    private String name;
    List<FamilyMember> children;

    public FamilyMember(String name) {
        this.name = name;
        children = new ArrayList<FamilyMember>();
    }

    public void addChildren(FamilyMember f){
        children.add(f);
    }

    public List<FamilyMember> getAllParents() {
        List<FamilyMember> parents = new ArrayList<FamilyMember>();
        if(children.size() > 0) {
            parents.add(this);
            for(int i = 0; i < children.size(); i++) {
                parents.addAll(children.get(i).getAllParents());
            }
        }
        return parents;
    }
}
```

Since each child is a **FamilyMember** type, we are able to call the method **getAllParents()** recursively, since the method adds all the elements to a list we are able to add it to the caller's list.

Let's demo this!

Memoization

Memoization is a technique for storing the results of a computation.

You may know it as a different term: *Caching*

We store the result as we may need to reuse it later.

Let's say we had a website that computes a simple page, the page doesn't differ between each user access, we could keep the result and send it every time it is asked.

So what does recursion and memoization have to do with each other?

Recursive call Tree

$F_n = F_{n-1} + F_{n-2}$

where $F_0=0$, $F_1=1$

n	0	1	2	3	4	5	6	7	8	9	...
F_n	0	1	1	2	3	5	8	13	21	34	
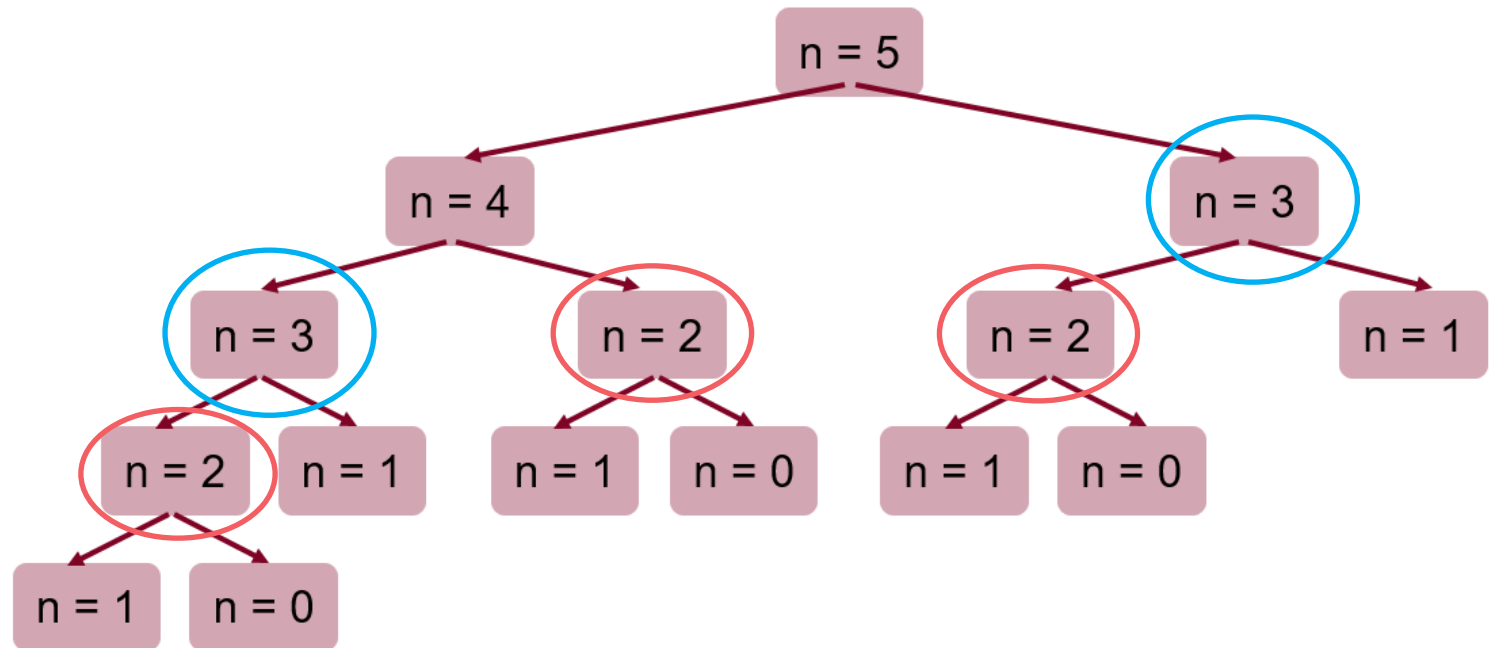
Memoization

Recursive calls can be computationally **expensive** and if we are repeatedly calling dependent values or the same values it makes sense to keep a record of that.

Simply, we are maintaining a copy of the answer because other computations depend on it.

Memoization

Welcome back to the fibonacci program!

```java
public class Fibonacci {



    public int[] generateSequence(int n) {



        if(n == 0) {
            return new int[] {0};
        } else if(n == 1) {
            return new int[] {0, 1};
        } else {
            int[] f1 = generateSequence(n-1);
            int[] f2 = generateSequence(n-2);
            int[] newF = new int[f1.length+1];
            for(int i = 0; i < f1.length; i++)
                newF[i] = f1[i];
            newF[newF.length-1] = f1[f1.length-1] + f2[f2.length-1];

            return newF;
        }
    }
}
```

So let's cache it..

Memoization

```java
public class FibonacciCache {



    public int[] generateSequence(int n) {



        if(n == 0) {
            return new int[] {0};
        } else if(n == 1) {
            return new int[] {0, 1};
        } else {
            int[] f1 = generateSequence(n-1);
            int[] f2 = generateSequence(n-2);
            int[] newF = new int[f1.length+1];
            for(int i = 0; i < f1.length; i++)
                newF[i] = f1[i];
            newF[newF.length-1] = f1[f1.length-1] + f2[f2.length-1];

            return newF;
        }
    }
}
```

Memoization

```java
public class FibonacciCache {
    private Map<Integer, int[]> cache;



    public int[] generateSequence(int n) {




        if(n == 0) {
            return new int[] {0};
        } else if(n == 1) {
            return new int[] {0, 1};
        } else {
            int[] f1 = generateSequence(n-1);
            int[] f2 = generateSequence(n-2);
            int[] newF = new int[f1.length+1];
            for(int i = 0; i < f1.length; i++)
                newF[i] = f1[i];
            newF[newF.length-1] = f1[f1.length-1] + f2[f2.length-1];

            return newF;
        }
    }
}
```

We've introduced a collection that will hold our answers. For convenience we are using a **Map**

Memoization

```java
public class FibonacciCache {
    private Map<Integer, int[]> cache;
    public FibonacciCache() {
        cache = new HashMap<Integer, int[]>();
    }
    public int[] generateSequence(int n) {



        if(n == 0) {
            return new int[] {0};
        } else if(n == 1) {
            return new int[] {0, 1};
        } else {
            int[] f1 = generateSequence(n-1);
            int[] f2 = generateSequence(n-2);
            int[] newF = new int[f1.length+1];
            for(int i = 0; i < f1.length; i++)
                newF[i] = f1[i];
            newF[newF.length-1] = f1[f1.length-1] + f2[f2.length-1];

            return newF;
        }
    }
}
```

Constructing it with an Integer as a key (the nth fibonacci sequence) and int[] as the value.

Memoization

```java
public class FibonacciCache {
    private Map<Integer, int[]> cache;
    public FibonacciCache() {
        cache = new HashMap<Integer, int[]>();
    }
    public int[] generateSequence(int n) {



        if(n == 0) {
            return new int[] {0};
        } else if(n == 1) {
            return new int[] {0, 1};
        } else {
            int[] f1 = generateSequence(n-1);
            int[] f2 = generateSequence(n-2);
            int[] newF = new int[f1.length+1];
            for(int i = 0; i < f1.length; i++)
                newF[i] = f1[i];
            newF[newF.length-1] = f1[f1.length-1] + f2[f2.length-1];
            cache.put(n, newF);
            return newF;
        }
    }
}
```

Once we generate a new list, we add it to the cache

Memoization

```java
public class FibonacciCache {
    private Map<Integer, int[]> cache;
    public FibonacciCache() {
        cache = new HashMap<Integer, int[]>();
    }
    public int[] generateSequence(int n) {
        if(cache.containsKey(n)) {
            return cache.get(n);
        } else {
            if(n == 0) {
                return new int[] {0};
            } else if(n == 1) {
                return new int[] {0, 1};
            } else {
                int[] f1 = generateSequence(n-1);
                int[] f2 = generateSequence(n-2);
                int[] newF = new int[f1.length+1];
                for(int i = 0; i < f1.length; i++)
                    newF[i] = f1[i];
                newF[newF.length-1] = f1[f1.length-1] + f2[f2.length-1];
                cache.put(n, newF);
                return newF;
            }
        }
    }
}
```

We have added a check to see if we have already computed this answer before, if we have we simply return it!

Let's demo this!

Take a break!

The University of Sydney

Documenting our work

Javadoc

Documentation is an important aspect to application development. You are producing a technical manual for others to read so they can comprehend your code and utilise it.

Providing a solution is not enough, you will need to show how to use the solution.

You will commonly work in teams and produce code that has to be readable by others.

Javadoc

- Comments

Simply writing comments in your code allows you to always understand what a method/class is doing.

Complex methods sometimes require inline details so users unfamiliar with your library can understand what it is trying to do.

- Clear method names and style

Try to ensure you adhere to the library's style guide and when writing your method name, make it clear to whoever is reading it what the method performs.

So what's javadoc?

Javadoc

Javadoc is a documentation generator for **Java**.

It extracts the **javadoc** comments for methods, fields and classes written within your java source files and produces a html documents.

The style is similar to the java api documentation.

Javadoc

Javadoc comments have a simple identifier but can contain annotations
that provide extra information about parameters and return types.

```
/**
* Given a target, a cat will attempt to pounce
* and attack it with its claws.
* If successful, this will return true, otherwise false
* @param t, A cat's enemy target
* @return success
*/
```

Javadoc

Javadoc comments have a simple identifier but can contain annotations that provide extra information about parameters and return types.

/**

> Simple identifier for a **javadoc** comment. Most **IDEs** will detect when you are writing a javadoc comment.

* Given a target, a cat will attempt to pounce

* and attack it with its claws.

* If successful, this will return true, otherwise false

* @param t, A cat's enemy target

* @return success

*/

Javadoc

Javadoc comments have a simple identifier but can contain annotations that provide extra information about parameters and return types.

```
/**
* Given a target, a cat will attempt to pounce
* and attack it with its claws.
* If successful, this will return true, otherwise false
* @param t, A cat's enemy target
* @return success
*/
```

@param, allows us to specify a parameter, specifies a what the parameter is, allows explanation

Javadoc

Javadoc comments have a simple identifier but can contain annotations that provide extra information about parameters and return types.

```
/**
* Given a target, a cat will attempt to pounce
* and attack it with its claws.
* If successful, this will return true, otherwise false
* @param t, A cat's enemy target
* @return success
*/
```

Similar to the @param annotation, @return outlines the return variable/value.

Javadoc

@param, parameter description, specified for as many parameters a method allows.

@see, reference annotation, useful for outlining issues with the method or what it has resolved.

@return, return type description. Specifies what is returned by the method and the conditions.

@since, when the method was included in your library

@throws, allows you to specify what exception it will throw and when it will throw it.

@deprecated, marks a method/class for deprecation (removal)

https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

So how do we generate documentation?

Javadoc

Let's consider the following class

```java
public class Cat {
  /**
   * The name of the cat
   */
  private String name;

  /**
   * Marks if the cat is in a playful state or not
   */
  private boolean playful;

  /**
   * New name counter, will be randomly assigned
   * when the name of the cat is changed
   */
  private int newNameCounter;

  /**
   * Old name of the cat
   */
  private String oldName;

  /**
   * Constructor for a cat, requires a name
   * default state for playful is false
   * @param name of the cat
   */
  public Cat(String name) {
    this.name = name;
    playful = false;
    newNameCounter = (int)(Math.random() * 30);
  }
}
```

```java
  /**
   * Given a target, a cat will attempt to pounce
   * and attack it with its claws.
   * If successful, this will return true, otherwise false
   * @param t, A cat's enemy target
   * @return success
   */
  public boolean attack(Target t) {
    if(target.isRodent()) { return true; } else { return false; }
  }

  /**
   * if the newNameCounter greater than 0, old name is return
   * otherwise newName is returned.
   * @return oldName or name,
   */
  public String getName() {
    if(newNameCounter > 0) {
      return oldName;
    } else {
      newNameCounter--;
      return name;
    }
  }
}
```

Javadoc

Let's consider the following class

```java
public class Cat {
  /**
   * The name of the cat
   */
  private String name;

  /**
   * Marks if the cat is in a playful state or not
   */
  private boolean playful;

  /**
   * New name counter, will be randomly assigned
   * when the name of the cat is changed
   */
  private int newNameCounter;

  /**
   * Old name of the cat
   */
  private String oldName;

  /**
   * Constructor for a cat, requires a name
   * default state for playful is false
   * @param name of the cat
   */
  public Cat(String name) {
    this.name = name;
    playful = false;
    newNameCounter = (int)(Math.random() * 30);
  }
```

```java
  /**
   * Given a target, a cat will attempt to pounce
   * and attack it with its claws.
   * If successful, this will return true, otherwise false
   * @param t, A cat's enemy target
   * @return success
   */
  public boolean attack(Target t) {
    if(target.isRodent()) { return true; } else { return false; }
  }

  /**
   * if the newNameCounter greater than 0, old name is return
   * otherwise newName is returned.
   * @return oldName or name,
   */
  public String getName() {
    if(newNameCounter > 0) {
      return oldName;
    } else {
      newNameCounter--;
      return name;
    }
  }
}
```

Specified a javadoc comment for a constructor, showing the @param name.

Javadoc

Let's consider the following class

```java
public class Cat {
    /**
     * The name of the cat
     */
    private String name;

    /**
     * Marks if the cat is in a playful state or not
     */
    private boolean playful;

    /**
     * New name counter, will be randomly assigned
     * when the name of the cat is changed
     */
    private int newNameCounter;

    /**
     * Old name of the cat
     */
    private String oldName;

    /**
     * Constructor for a cat, requires a name
     * default state for playful is false
     * @param name of the cat
     */
    public Cat(String name) {
        this.name = name;
        playful = false;
        newNameCounter = (int)(Math.random() * 30);
    }
}
```

```java
    /**
     * Given a target, a cat will attempt to pounce
     * and attack it with its claws.
     * If successful, this will return true, otherwise false
     * @param t, A cat's enemy target
     * @return success
     */
    public boolean attack(Target t) {
        if(target.isRodent()) { return true; } else { return false; }
    }

    /**
     * if the newNameCounter greater than 0, old name is return
     * otherwise newName is returned.
     * @return oldName or name,
     */
    public String getName() {
        if(newNameCounter > 0) {
            return oldName;
        } else {
            newNameCounter--;
            return name;
        }
    }
}
```

We specify the parameter and a return description.

Javadoc

The javadoc command line tool requires us to specify a directory where the documents will be generated and point to our source files.

> javadoc -d <directory> <path to .java files>

Javadoc

Directories containing subpackages require the use of the **subpackages** flag to recursively search all .java files and map each package in the document.

```
> javadoc -d <directory> -subpackages packageName
```

Javadoc

Output of the following command will produce a directory called
cat_docs which will contain the generated documentation files.

```
> javadoc -d cat_docs Cat.java
```

./cat_do
cs

Javadoc

Class Cat

java.lang.Object
    Cat

```
public class Cat
extends java.lang.Object
```

Cat class, can represent many different kinds of cats such as Garfield, Felix, Maru and Grumpy Cat

Constructor Summary

Constructors

Constructor	Description
Cat(java.lang.String name)	Constructor for a cat, requires a name default state for playful is false

Method Summary

All Methods	Instance Methods	Concrete Methods

Modifier and Type	Method	Description
java.lang.String	**getName**()	if the newNameCounter greater than 0, old name is return otherwise newName is returned.
void	**meow**()	Meows, depending on the state it is in, it may be a happy meow or an unhappy meow
void	**setName** (java.lang.String name)	Sets a new name for the cat, may take a while for it to listen.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

We have produced our Cat class documentation.

Showing the documentation above the class.

Javadoc

SEARCH: 🔍 Search ✕

Class Cat

java.lang.Object
    Cat

```
public class Cat
extends java.lang.Object
```

Cat class, can represent many different kinds of cats such as Garfield, Felix, Maru and Grumpy Cat

We have produced our Cat class documentation.

Constructor Summary

Constructors

Constructor	Description
Cat(java.lang.String name)	Constructor for a cat, requires a name default state for playful is false

Showing the documentation above the class.

Method Summary

All Methods   **Instance Methods**   **Concrete Methods**

Modifier and Type	Method	Description
java.lang.String	getName()	if the newNameCounter greater than 0, old name is return otherwise newName is returned.
void	meow()	Meows, depending on the state it is in, it may be a happy meow or an unhappy meow
void	setName (java.lang.String name)	Sets a new name for the cat, may take a while for it to listen.

The documentation for each method and their description.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Let's go through an example

See you next time!

THE UNIVERSITY OF
SYDNEY