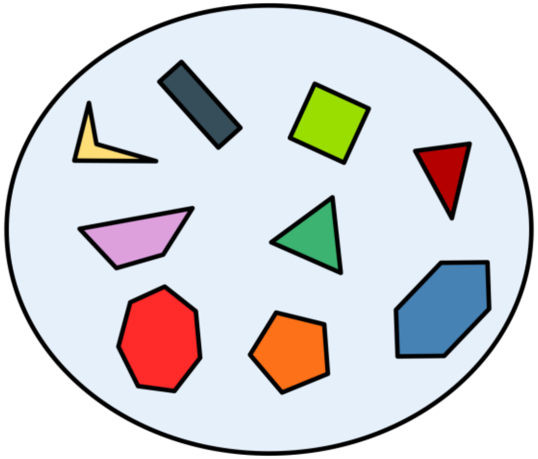
Module 1
Abstract Data Types (ADTs)

Abstractions

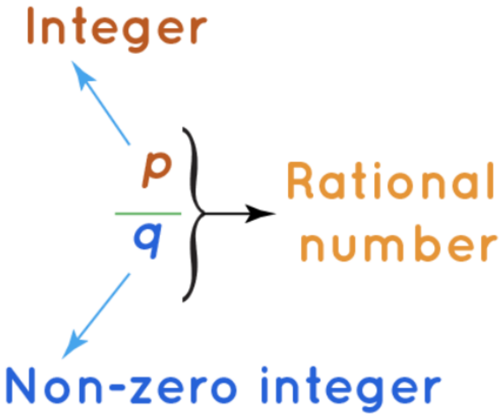
- In Mathematics, abstraction is the process of extracting underlying structures, patterns, or properties of a mathematical concept with the goal of generalizing to support wider applications

Abstractions

- Examples of abstractions in math:



Sets



$$P(A) = \frac{n}{N} = \frac{\#\,outcomes\ in\ A}{\#\,outcomes\ in\ Sample\ Space}$$

Abstractions

- Similarly, in CS, abstraction is the process of creating a simplified model of a system with the goal of solving a problem using computers

Abstractions

- To create an abstract model, we begin by removing unnecessary details and focus on key parts of a system relevant to a particular problem

Abstractions

- Programming Languages (PLs) support the creation of abstractions so we can write programs to solve problems

Abstractions

- Examples of abstractions in PLs are variables, expressions, conditional and iterative control structures, functions, recursion, data typing, encapsulation, and inheritance

ADTs

- Abstract Data Types (ADTs) refer to the ability to create data types to represent things and concepts
- A data type defines a collection of data values and a set of predefined operations on those values

ADTs

- Each PL has its own set of pre-defined (built-in) data types and a mechanism for the creation of (new) data types

ADTs

- For example, Java's built-in data types provide an efficient way to declare variables to represent numbers (byte, short, int, long, float, double), single characters (char), and true/false (boolean)

ADTs

- The most common mechanism for the creation of data types is the class
- Classes are available in PLs that support the object-oriented programming (OOP) paradigm

ADTs

- A class puts together in a single entity values (represented by fields) and functionality (represented by methods)
- Fields represent attributes, properties, and states, while methods are associated with code

ADTs

- Consider a <u>Rectangle</u> class that models a quadrilateral with four right angles
- A rectangle can be defined by its <u>length</u> and <u>width</u>
- Given a rectangle we are interested in computing its <u>area</u>, <u>perimeter</u>, and <u>diagonal</u>.

```
class Rectangle {

   // member variables (a.k.a. fields)
   int width;
   int height;

   // constructors
   Rectangle(int width, int height) {
     if (width <= 0)
       this.width = 1;
     else
       this.width = width;
     this.height = height <= 0 ? 1 : height;
   }

   Rectangle() {
     width = height = 1;
   }

   // methods
   double area() {
     return width * height;
   }

   double perimeter() {
     return 2 * width + 2 * height;
   }

   double diagonal() {
     return Math.sqrt(width * width + height * height);
   }
 }
```

A rectangle class definition

```
class Rectangle {

  // member variables (a.k.a. fields)
  int width;
  int height;

  // constructors
  Rectangle(int width, int height) {
    if (width <= 0)
      this.width = 1;
    else
      this.width = width;
    this.height = height <= 0 ? 1 : height;
  }

  Rectangle() {
    width = height = 1;
  }

  // methods
  double area() {
    return width * height;
  }

  double perimeter() {
    return 2 * width + 2 * height;
  }

  double diagonal() {
    return Math.sqrt(width * width + height * height);
  }
}
```

fields

A rectangle class definition

```
class Rectangle {

  // member variables (a.k.a. fields)
  int width;
  int height;

  // constructors
  Rectangle(int width, int height) {
    if (width <= 0)
      this.width = 1;
    else
      this.width = width;
    this.height = height <= 0 ? 1 : height;
  }

  Rectangle() {
    width = height = 1;
  }

  // methods
  double area() {
    return width * height;
  }

  double perimeter() {
    return 2 * width + 2 * height;
  }

  double diagonal() {
    return Math.sqrt(width * width + height * height);
  }
}
```

fields

A rectangle class definition

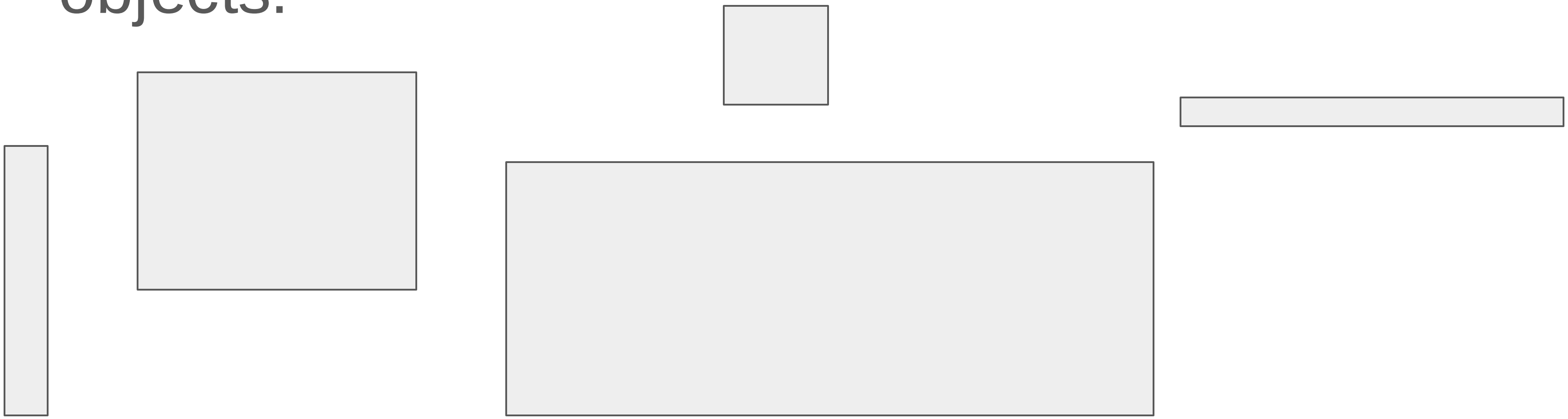
methods

METROPOLITAN STATE UNIVERSITY℠ OF DENVER

LIVES TRANSFORMED

Objects

- From a class you can create as many objects as you want
- Below are visual representations of rectangle objects:



METROPOLITAN STATE UNIVERSITY OF DENVER
LIVES TRANSFORMED

Objects

- Objects are created from classes in a process called “object instantiation”
- In Java we use the “new” operator for “object instantiation”
- Examples:

```
new Rectangle(10, 5)

new Rectangle(2, 8)
```

Objects

- After its instantiation, an object is created in *some* location in the computer's memory
- In order for an object to be accessed in programs its "reference" needs to be saved in variables

Objects

- For example:

```
Rectangle a = new Rectangle(10, 5)

Rectangle b = new Rectangle(2, 8)
```

- Once you have saved the references for your objects you can start using them

Objects

- For example:

```
Rectangle a = new Rectangle(10, 5)

Rectangle b = new Rectangle(2, 8)

System.out.println("Area of a is " +
a.area());

System.out.println("Perimeter of b is " +
b.perimeter());
```

Objects

- The object used to call a method is referred to as the *callee* object
- Example:

`a.area( )` a is the callee

Objects

- Within a method, any instance variables associated with the *callee* can be accessed directly
- In the previous example, the *area* method was called without informing the width and the height of the rectangle because the *callee* has access to those variables

Class Definition

- Variables defined within a class but outside any method are called member variables
- Member variables (also called fields, attributes, properties) can be instance variables or class variables

Class Definition

- Instance Variables:
  - There is one for each instance
  - In other words, each object has its own copy of instance variables

Class Definition

- Class variables:
  - There is one copy for ALL instances of the class
  - In other words, objects share class variables
  - To define a class variables in Java add the static keyword

Class Definition

- Consider a class to model eggs
- It makes sense to define the following as instance variables in an Egg class:

```
private int size;
private boolean freeRange;
```

Class Definition

- It also makes sense to define the following (constants) as class variables in the same Egg class:

```
private static final int SMALL       = 1;
private static final int MEDIUM      = 2;
private static final int LARGE       = 3;
private static final int EXTRA_LARGE = 4;
```

Class Definition

- Use Class Variables:
  - When defining constants
  - Constants in Java are defined using the keyword final
  - When you need to share information between instances:
    - For example: a global counter

Class Definition

- Functions defined within a class (scope) are called methods
- They can be <u>instance</u> or <u>class</u> methods
- Constructors are special methods that are used to initialize (create) objects
- Methods can have parameters and a return value

Class Definition

- Constructors are special methods that are used during instantiation only
- In Java, they have the same name of the class and they are called with the use of the *new* operator
- Constructors implicitly return a reference to the object created

METROPOLITAN STATE UNIVERSITY℠ OF DENVER
LIVES TRANSFORMED

Class Definition

- Class templates refer to a feature present in most PLs that allows parameterizing a class definition
- In Java, class templates are referred to as *generics* (or generic classes)

Class Definition

- Imagine that you want to create a class definition for a box that can hold a single object of a specific type

Class Definition

```
public class Box<T> {
    private T   content;
    private int length, width, height;

    public Box(T content, int length, int width, int height) {
        this.content = content;
        this.length  = length;
        this.width   = width;
        this.height  = height;
    }

    public T getContent() { return content; }
    public getLength()    { return length;  }
    public getWidth()     { return width;   }
    public getHeight()    { return height;  }
}
```

Interface Definition

- Interfaces are ADTs used to specify a behavior (set of abstract methods) that a class must implement
- An abstract method is a method without body (code)

Interface Definition

- The Java STD (Standard Library) comes with pre-defined interfaces for specific uses
- An example is the Comparable interface which is useful when objects of a particular class need to be compared

Interface Definition

- The Comparable interface is a generic interface which means that to use it you need to specify the type of objects being compared

```
public interface Comparable<T> {
    public int compareTo(T other);
}
```

Interface Definition

- compareTo returns:
  - a negative integer, if the *callee* is less than the given object;
  - zero, if the *callee* is equal to the given object;
  - a positive integer, if the *callee* object is greater than given object.

Recursion

- Another important abstraction in CS is recursion
- Recursion occurs when you have a definition that uses itself
- For example, a function (or a class) can be defined in terms of itself

Recursion

- Consider the factorial definition of a non-negative integer number n:

```
n! = n x (n-1) x … x 1
```

Recursion

- The factorial of n can also be defined in terms of itself:

```
n! = n * (n - 1)!
```

- The recursive definition above is incorrect because the computation goes on forever as n decrements

Recursion

- Below is the correct recursive definition for the factorial of a non-negative integer:

```
n! = n * (n - 1)!, if n > 0
      n! = 1, if n = 0
```

base case or exit condition

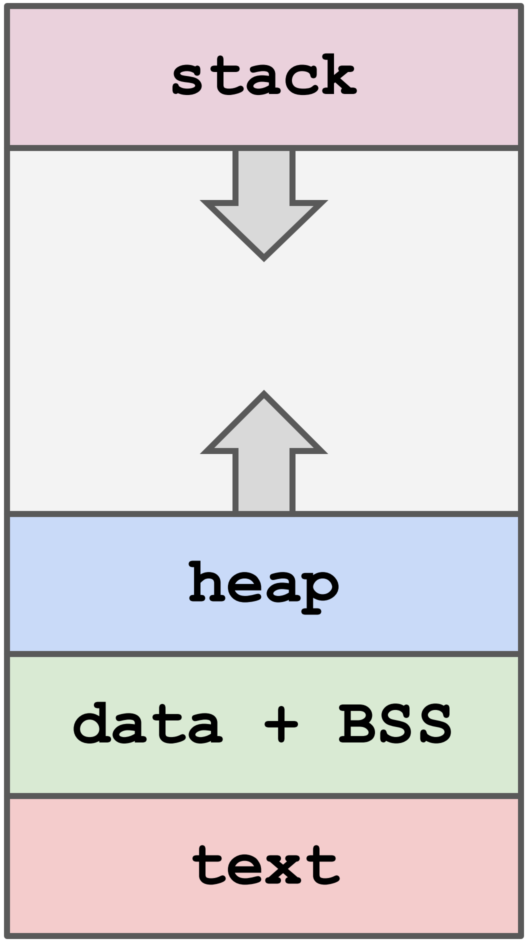
Recursion

- Below is an implementation of a recursive procedure to compute the factorial of a given number:

```
static int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial( n: n - 1);
}
```

Recursion

- The call stack is an internal data structure used by the O.S. to keep track of function calls in a program
- Other names: execution stack, run-time stack, program stack, control stack



Memory Segments of
a Running Program

Recursion

- Its main purpose is to keep track of the point to which each active function should return control when it finishes executing
- The call stack is made of “stack frames”

Recursion

- A new “stack frame” is created and pushed onto the call stack every time your program makes a function call
- Conversely, a “stack frame” is popped out of the call stack whenever a function in your program resumes execution (i.e. return)

Recursion

factorial(4) = ?

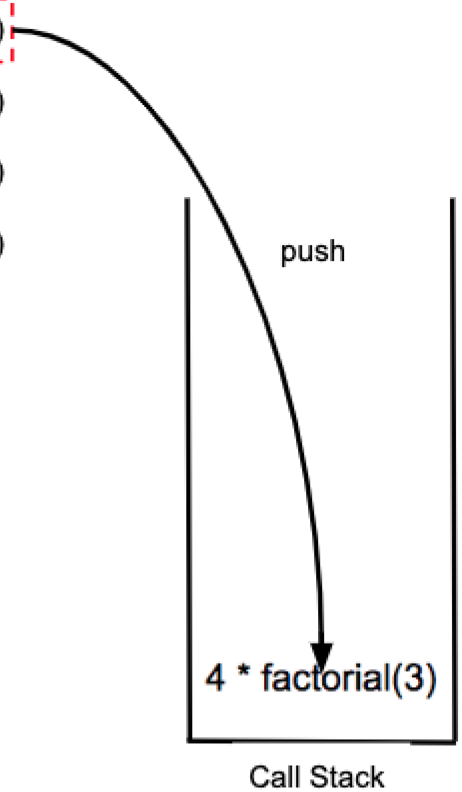
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * factorial(0)

factorial(0) = 1 (base case)



Recursion

factorial(4) = ?

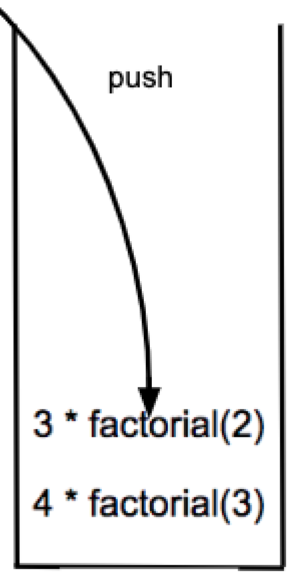
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * factorial(0)

factorial(0) = 1 (base case)



Call Stack

Recursion

factorial(4) = ?

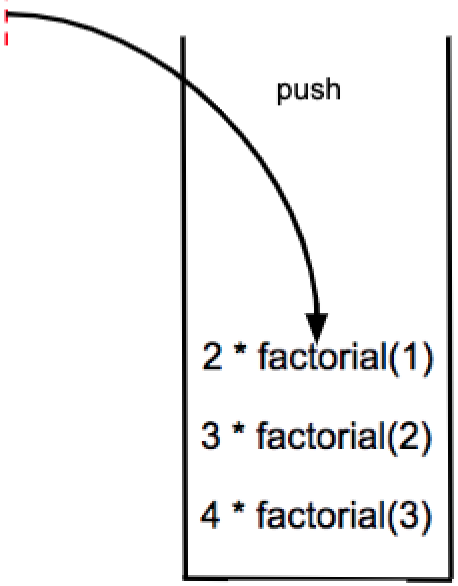
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * factorial(0)

factorial(0) = 1 (base case)



Call Stack

Recursion

factorial(4) = ?

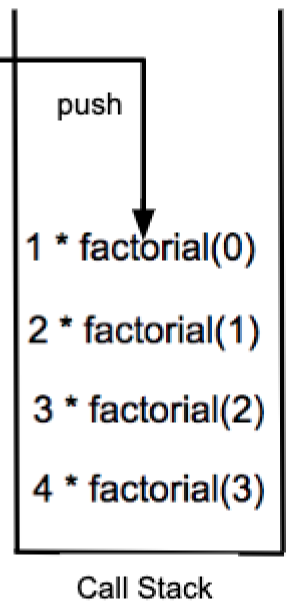
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * factorial(0)

factorial(0) = 1 (base case)



Recursion

factorial(4) = ?

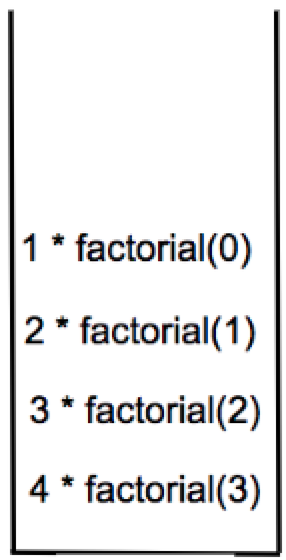
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * factorial(0)

factorial(0) = 1 (base case)



Call Stack

Recursion

factorial(4) = ?

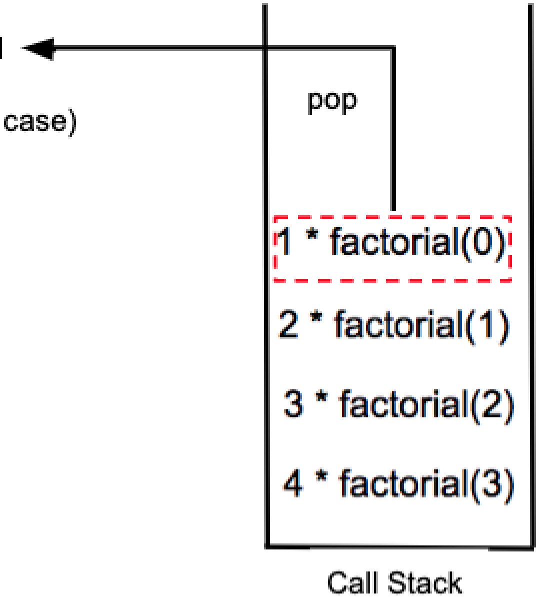
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * 1 = 1

factorial(0) = 1 (base case)



Recursion

factorial(4) = ?

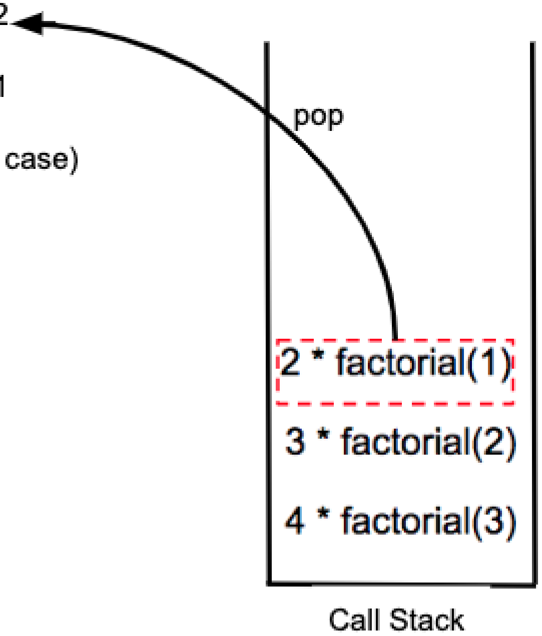
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * 1 = 2

factorial(1) = 1 * 1 = 1

factorial(0) = 1 (base case)



Recursion

factorial(4) = ?

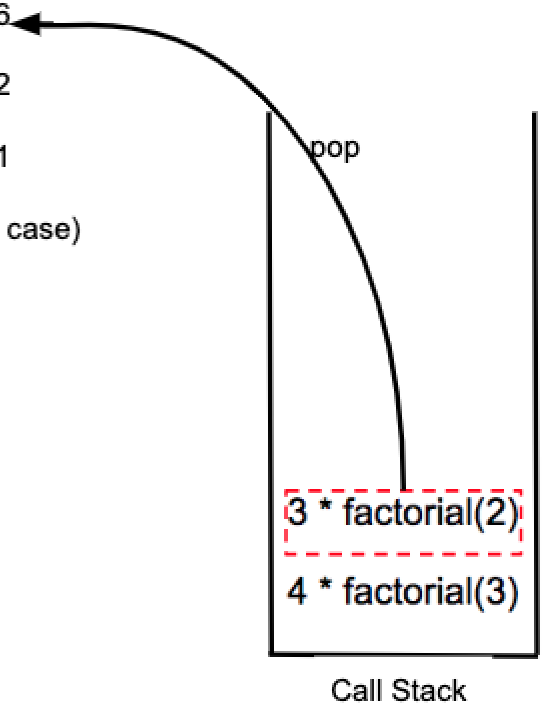
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * 2 = 6

factorial(2) = 2 * 1 = 2

factorial(1) = 1 * 1 = 1

factorial(0) = 1 (base case)



Recursion

factorial(4) = ?

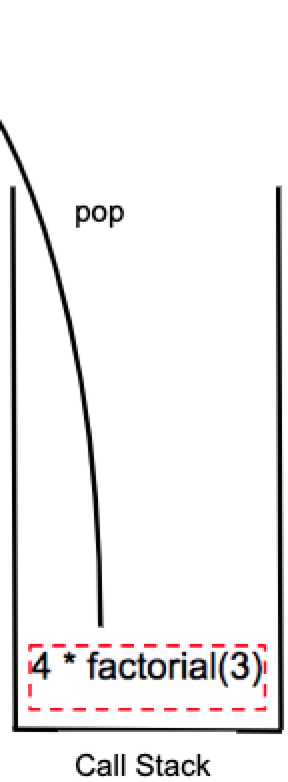
factorial(4) = 4 * 6 = 24

factorial(3) = 3 * 2 = 6

factorial(2) = 2 * 1 = 2

factorial(1) = 1 * 1 = 1

factorial(0) = 1 (base case)



Recursion

- The "stack overflow" error occurs when the call stack is exhausted because of the number of function calls made recursively without the function ever reaching its base case

Recursion

- Another application of recursion is to define ADTs
- PLs like Java allow the recursive definition of classes

Recursion

- Consider the recursively defined Node class below:

```
public class Node<E> {
    E       value;
    Node<E> next;

    public Node(E value) {
        this.value = value;
        next = null;
    }
}
```

Definitions like the Node class will be very useful in creating more advanced ADTs

Inheritance

- Some classes share member variables (or functionality) that are similar
- In these cases, it does not make sense to have duplicated definitions
- Instead, we should abstract things that are common and define a parent class to model them

Inheritance

- Let’s say that you want to model different types of motor vehicles, including motorcycles, cars, trucks, and buses
- You can begin your design by defining a class to model a generic motor vehicle with the things that are common to all types of vehicles

Inheritance

- All motor vehicles are built on a particular year by a manufacturer, and they all have a specific model name designation
- Consider the definition for a Vehicle class according to what was described

Inheritance

```java
class Vehicle {

    private String manufacturer;
    private String model;
    private int year;

    Vehicle(String manufacturer, String model, int year) {
        this.manufacturer = manufacturer;
        this.model = model;
        this.year = year;
    }

    // ...
}
```

Inheritance

- For some types of motor vehicles, we might be interested in capturing specific features
- For example, let's say we want to classify passenger cars by type using designations such as sedan, station wagon, or hatchback

Inheritance

- Instead of creating a new class for passenger cars, the best approach is to use inheritance which allows the definition of a class in terms of another
- To use inheritance in Java we use the keyword *extends* when defining a class

Inheritance

```
class Car extends Vehicle {

    private String type;

    Car(String manufacturer, String model, int year, String type) {
        super(manufacturer, model, year);
        this.type = type;
    }

    // ...

}
```

Inheritance

- Similarly, for trucks we might be more interested in their class number, a number from 1 to 8 based on the truck’s gross vehicle weight rating (GVWR)

Inheritance

```java
class Truck extends Vehicle {

    static final int DEFAULT_CLASS = 1;
    private int classification;

    Truck(String manufacturer, String model, int year, int classification) {
        super(manufacturer, model, year);
        if (classification > 0 && classification < 9)
            this.classification = classification;
        else
            this.classification = DEFAULT_CLASS;
    }

    // ...

}
```

Inheritance

- Inheritance is a powerful abstraction present in object-oriented languages such as Java
- It makes your code much easier to maintain because it reduces code repetition by encouraging code reuse whenever it is possible

Inheritance

- All classes in Java automatically inherit from the Object class
- Therefore, methods defined in the object class are available in all classes in Java, including: `toString`, `equals`, `clone`, and `hashCode`

Inheritance

- Overriding (or *method override*) refers to the redefinition of an inherited method
- It is a good practice in Java to explicitly declare an override using the annotation @override

Inheritance

- The most important method overrides in Java are `toString`, `equals`, and `clone`
- It is a good practice to override at least the `toString` and `equals` methods in your own class definitions

Inheritance

- `toString` returns a string representation of an object
- `equals` allows structure comparison of objects of a specific type
- `clone` returns a new object that is an exact copy of the *callee* object

Encapsulation

- Another important abstraction in CS is encapsulation which is a way to protect access to variables and methods in a class
- Java offers 4 access level modifiers for variables and methods:
  private, (default), *protected*, and *public*

Encapsulation

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

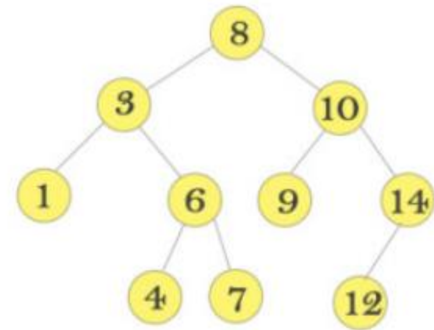
Source: http://net-informations.com

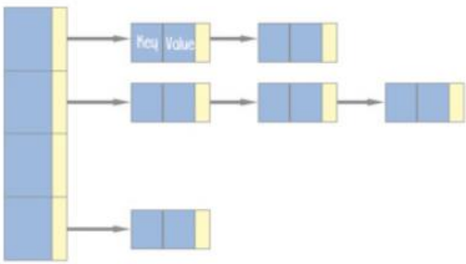
METROPOLITAN STATE UNIVERSITY OF DENVER
LIVES TRANSFORMED

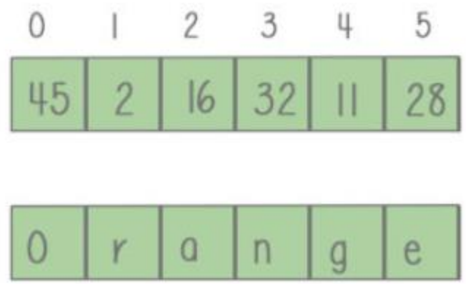
Data Structures

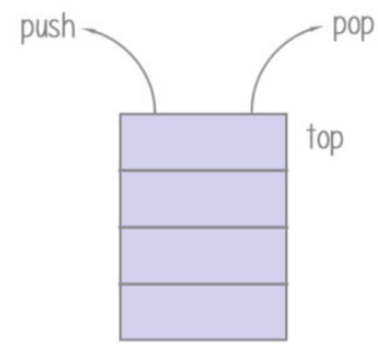
- Previously we defined a class as the most important mechanism for abstract data typing in object-oriented PLs such as Java
- Because of their importance in so many different applications, powerful and general-purpose classes called <u>data structures</u> are studied extensively in CS

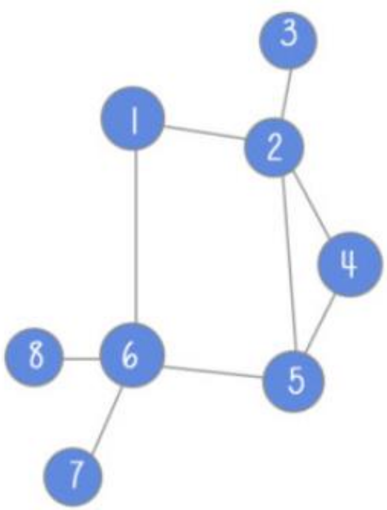
Data Structures

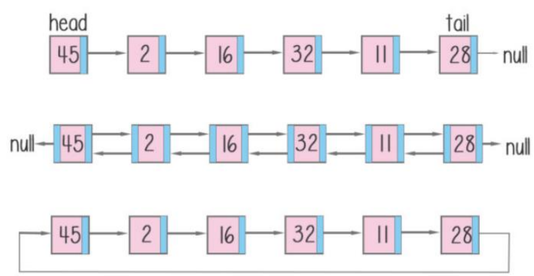












Software Development Testing

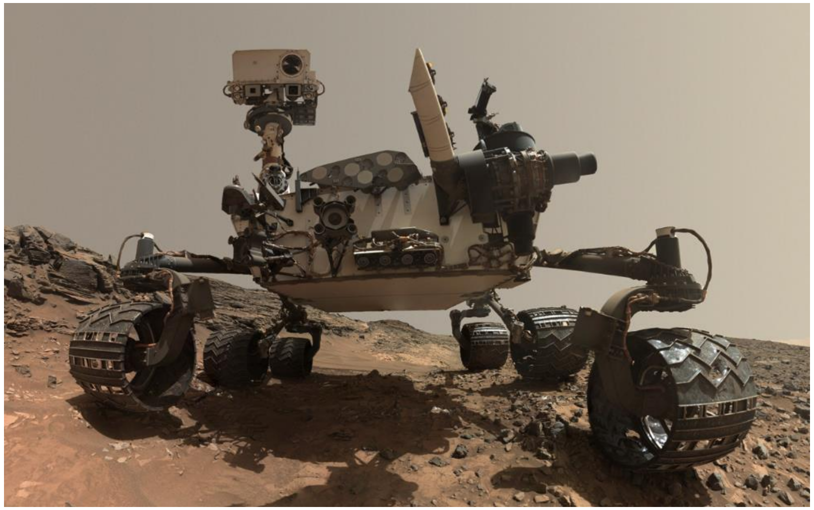
- ADTs must work according to their specifications
- Bugs in software can have catastrophic consequences
- To avoid bugs ADT implementations must be tested systematically

Software Development Testing

- Ariane 5 Failure (1996):
  - Rocket exploded just 40s after its lift-off
  - $7 billion dollar project
  - Caused by a 64-bit floating-point number converted to a 16-bit signed integer

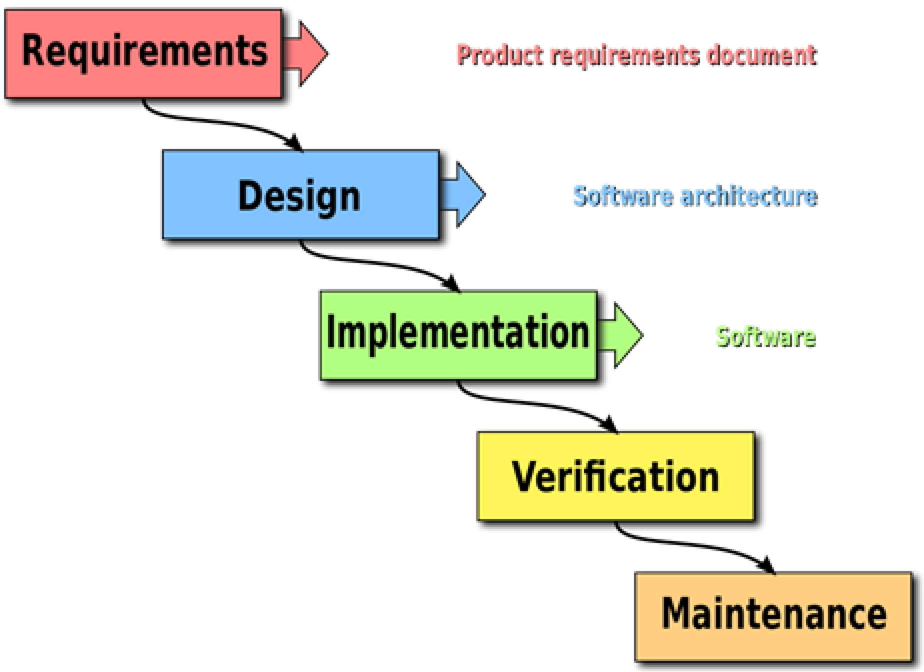
Software Development Testing

- Mars Curiosity's out-of-memory problem was only found after the rover was already on Mars in 2013



Software Development Testing

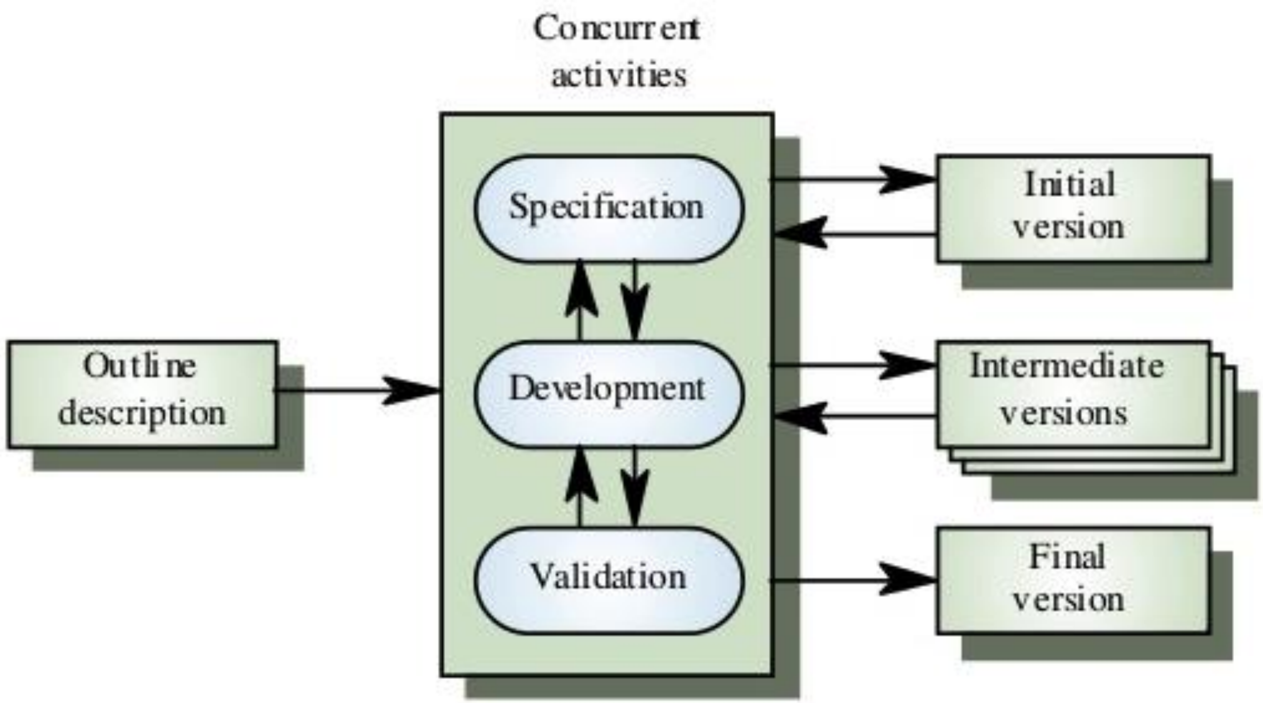
- The OLD Waterfall Model:



Software Development Testing

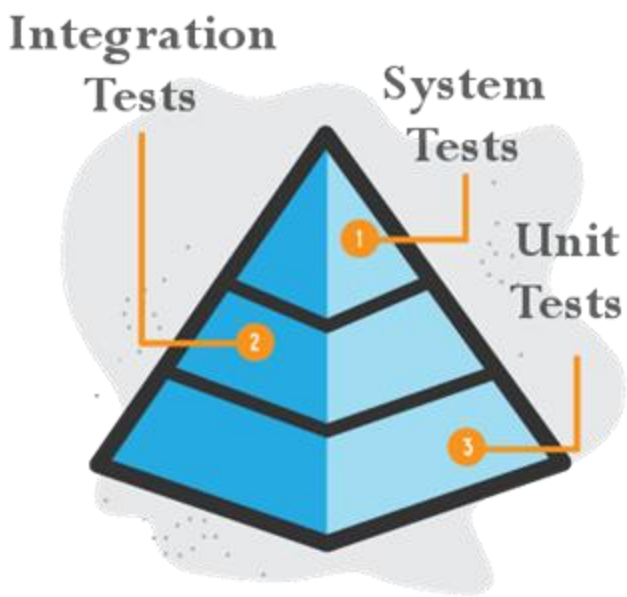
- Deliver working software incrementally in short iterations
- Get working software in front of real users as soon as possible
- Software is delivered in multiple versions until the product reaches user satisfaction
- Software is constantly being changed and updated:
    - Crucial that each version passes validation procedures

Software Development Testing



Software Development Testing

- Testing Levels:

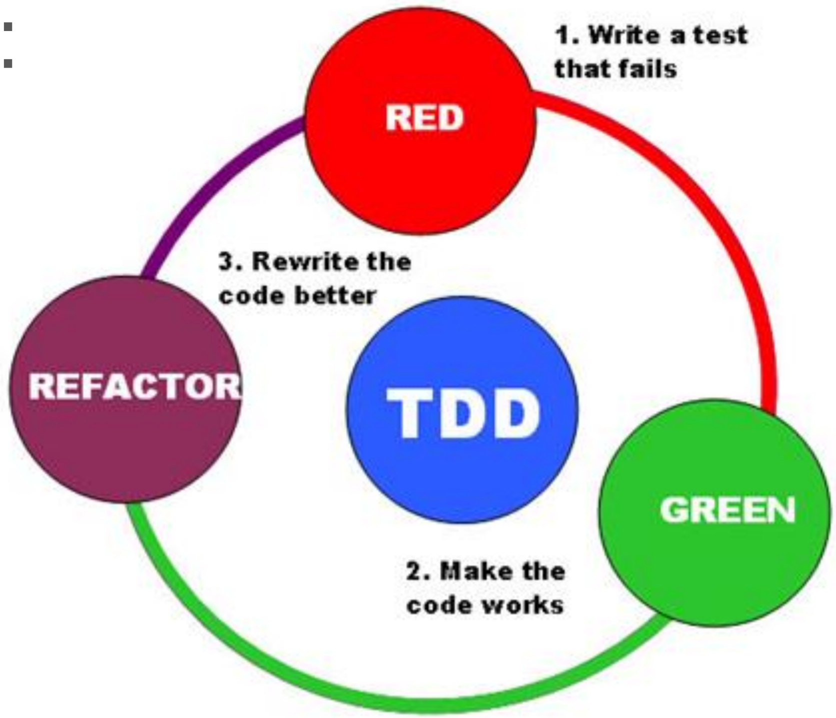


Software Development Testing

- Unit Testing:
  - Individual units of software code (function, class, etc.) are tested
  - The goal is to isolate each part of the program and show that the individual parts are correct
  - Unit testing can be automated using testing frameworks like JUnit in Java

Software Development Testing

- Unit Testing:
  - Best Practices:



Software Development Testing

- Best Practices:
  - Give your tests names as descriptive as possible
  - Describe the inputs and pre-conditions that must be satisfied before the test begins
  - Describe the expected outputs and post-conditions that must be satisfied after the procedure is executed

Software Development Testing

- Best Practices:
  - Use the scientific method:
    - Formulate hypothesis
    - Test the hypothesis
  - Make your tests as atomic as possible
  - Avoid test interdependence

Software Development Testing

- Unit Testing:
  - Floating-point Approximation Errors:

```java
double a = 0;
for (int i = 0; i < 10; i++)
    a += 0.1;

double b = 1;

System.out.println("Comparison using equality operator:");
if (a == b)
    System.out.println("a == b");
else
    System.out.println("a != b");
```

Software Development Testing

- Unit Testing:
  - Floating-point Approximation Errors:

```java
double a = 0;
for (int i = 0; i < 10; i++)
    a += 0.1;

double b = 1;

System.out.println("\nComparison accepting some deviation:");
if (Math.abs(a - b) <= 0.001)
    System.out.println("a == b");
else
    System.out.println("a != b");
```

Software Development Testing

- Unit Testing:
  - Floating-point Approximation Errors:

```
double x = 1 / 3.;
double y = x + 1 - 1;
System.out.println("x = " + x);
System.out.println("y = " + y);
if (x == y)
    System.out.println("x == y");
else
    System.out.println("x != y");
```

Documentation

- ADTs must be documented so people know how and which scenarios to use them
- At a minimum, an ADT implementation can be documented using comments
- Java defines special syntax for documentation in comments called Java Docs