# Functions (Static Methods)

- **Java function.**
  - Takes zero or more input arguments.
  - Returns one output value.

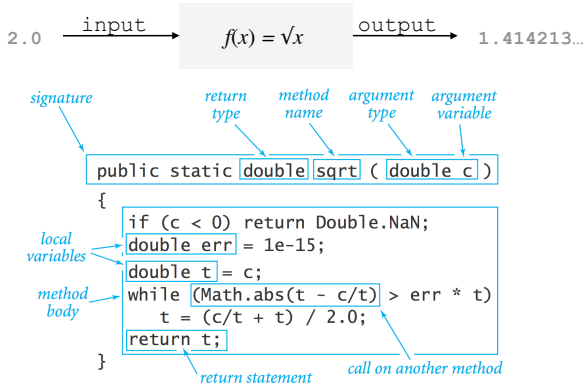
- **Applications.**
  - Scientists use mathematical functions to calculate formulas.
  - Programmers use functions to build modular programs.
  - You use functions for both.

- **Examples.**
  - Built-in functions: `Math.random()`, `Math.abs()`, `Integer.parseInt()`.
  - Our I/O libraries `StdDraw.show()`, `StdAudio.play()`.
  - User-defined functions: `main()`.



# Anatomy of a Java Function

- **Java functions.** Easy to write your own.






# Flow of Control

- **Flow of control.** Functions provide a new way to control the flow of execution of a program.

```
public class Newton
{
   public static double sqrt(double c)
   {
      if (c < 0) return Double.NaN;
      double err = 1e-15;
      double t = c;
      while (Math.abs(t - c/t) > err * t)
         t = (c/t + t) / 2.0;
      return t;
   }

   public static void main(String[] args)
   {
      int N = args.length;
      double[] a = new double[N];
      for (int i = 0; i < N; i++)
         a[i] = Double.parseDouble(args[i]);
      for (int i = 0; i < N; i++)
      {
         double x = sqrt(a[i]);
         StdOut.println(x);
      }
   }
}
```

"pass-by-value"

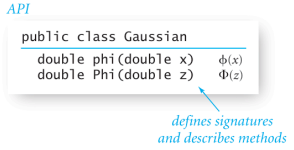


# Libraries

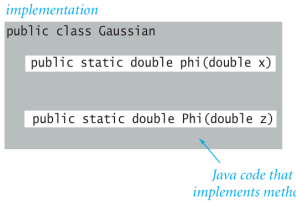
- **Library.** A module whose methods are primarily intended for use by many other programs.


- **Client.** Program that calls a libra

- **API.** Contract between client and implementation.




- **Implementation.** Program that implements the methods in an A



## Modular Programming

- Modular programming.
  - Divide program into self-contained pieces.
  - Test each piece individually.
  - Combine pieces to make program.

- Ex. Flip N coins. How many heads?
  - Read arguments from user.
  - Flip one fair coin.
  - Flip N fair coins and count number of heads.
  - Repeat simulation, counting number of times each outcome occurs.
  - Plot
  - Com

```
% java Bernoulli 20 100000
```





## Flashback: Data processing

- Scan a large (~ $10^7$ bytes) file
- Print the words together with counts of how often they occur
- Need more specification?

- How do you do it?

- What is we only wanted the top $k$ (say 20) words?



## What can you put into an ArrayList?

- Any `Object`
- Use a wrapper class (see java.lang.*)
  - `int, double, char, boolean, ...`
  - `Integer, Double, Character, Boolean,`
- Can have your cake and eat it too

```
ArrayList<Integer> list = new ArrayList<Integer>();
for (int k = 0; k < 10; k++){
     list.add(k*k);
 }
 for (Integer jj : list){
     System.out.println(jj);
 }
```

- All made practical by Version 5 of Java



## Exploring ArrayLists

- Look at the Java 6 API
- Note interfaces implemented
  - Serializable, Cloneable, Iterable
  - Collection, List, RandomAccess
- Note other descriptive text
  - Regarding performance
  - Constructors
  - Methods
  - Don't forget methods in parent classes

## Exploring ArrayLists

- Some Commonly Used Methods
  - `boolean add(E o)` *// append*
  - `void add(int index, E element)` *// insert*
  - `void clear()`
  - `boolean contains(Object elem)`
  - `E get(int index)`
  - `int indexOf(Object elem)`
  - `boolean remove(Object o)`
  - `E remove(int index)`
  - `E set(int index, E elem)` *// replace*
  - `int size()`



## Exploring ArrayLists

- Performance
  - Constant Time
    - size, isEmpty, get, set, iterator, listIterator operations
    - add (amortized)
  - Linear Time
    - All of the other operations run in linear time
  - What does all of this mean?
  - Why do we care?
  - Exercise: Implement on an array the equivalent of
    - `void add(int index, E element)`
    - `E remove(int index)`
  - Remember: Memory is an array (well sort of)



## What is a char?

- Differences between unicode and ASCII
  - Why is unicode used? Why should we care? What should we know? How many of the details are important?
- A char value can be treated like an int value
  - Add integer to it, cast back to char
  - Subtract character from it, get int back

```
counters[s.charAt(k)- 'A']++;
```

  - Anatomy of the statement above??



## Inheritance and Interfaces

- Inheritance models an "is-a" relationship
  - A dog is a mammal, an ArrayList is a List, a square is a shape, ...
- Write general programs to understand the abstraction, advantages?

```
void execute(Pixmap target) {
   // do something
}
```

- But a dog is also a quadruped, how can we deal with this?

## Single inheritance in Java

- A class can extend only one class in Java
  - All classes extend Object --- it's the root of the inheritance hierarchy tree
  - Can extend something else (which extends Object), why?
- Why do we use inheritance in designing programs/ systems?
  - Facilitate code-reuse (what does that mean?)
  - Ability to specialize and change behavior
    - If I could change how method `foo()` works, `bar()` is ok
  - Design methods to call ours, even before we implement
    - Hollywood principle: don't call us, …



## Comparable and Comparator

- Both are interfaces, there is no default implementation
  - Contrast with `.equals()`, default implementation?
  - Contrast with `.toString()`, default?
- Where do we define a Comparator?
  - In its own .java file, nothing wrong with that
  - Private, used for implementation and not public behavior
    - Use a nested class, then decide on static or non-static
    - Non-static is part of an object, access inner fields
- How do we use the Comparator?
  - Sort, Sets, Maps (in the future)
- Does hashing (future topic) have similar problems?



## Sets

- Set is an unordered list of items
  - Items are unique! Only one copy of each item in set!
- We will use two different implementations of sets
- `TreeSet`
  - A TreeSet is backed up by a tree structure (future topic)
  - Keeps items sorted (+)
  - Slower than HashSets ?? (-)
- `HashSet`
  - A HashSet is backed up by a hashing scheme (future topic)
  - Items not sorted – should seem to be in random order (-)
  - Faster than TreeSets ?? (+)



## Using Both ArrayList and Sets

- You may want to use a set to get rid of duplicates, then put the items in an ArrayList and sort them!
- Problem:
  - Often data comes in the form of an array
  - How do we go from array to ArrayList or TreeSet?
- Problem:
  - Often we are required to return an array
  - How do we go from a Collection such as an ArrayList or TreeSet to an array?
- Can do it the "hard" way with loops or iterators:
  - one item at a time
- OR: