

General Languages

Classification:

- Abstraction
 - Declarative
 - * Functional
 - Imperative
 - * Procedural
- Behavior
 - Dynamic
 - Static

Declarative Languages:

- not imperative
- describes what computation should be performed and not how to compute it
- lacks side effects (referentially transparent)
 - an expression always evaluates to the same result in any context
 - instance can be replaced with its corresponding value without changing the program's behavior
- clear correspondence to mathematical logic

Functional Languages:

- Def
 - define programs and subroutines as mathematical functions
 - many functional languages are "impure", containing imperative features
 - many functional languages are tied to mathematical calculation tools
 - declarative: programming is done with expressions or declarations instead of statements
 - the output value of a function depends only on the arguments that are input to the function
 - * calling a function f twice with the same value for an argument x will produce the same result $f(x)$ each time

- * eliminating side effects(changes in state that do not depend on the function inputs)
makes it much easier to understand and predict the behavior of a program

- Pure ex: Haskell
- Impure ex: SML

Imperative Languages:

- Def
 - uses statements that change a program's state

Scripting Languages:

Hierarchy of programming languages:

- High-Level language (C, Java, PHP, Python)
 - more complex than machine code
- Assembly language
 - machine code with names substituted for numbers
- Machine code
 - only numbers
- Hardware
- convert program into machine language
 - compile the program
 - interpret the program

Web Development Languages:

- HTML/XHTML
 - Defines content of web page
- CSS

- appearance of web page
- HTML+CSS can create static web pages
 - static pages can interact with your visitor through the use of forms
 - form is fill, request submitted and sent back to the server, new static web page is constructed and downloaded into the browser
 - disadvantage of static web pages, only way visitor interact with the page is by filling out the form and waiting for a new page to load
- javascript
 - behavior of web page
 - can validate each of the fields as visitor enter and provide immediate feedback when they make a typo (vs after they filled everything and submitted)
 - add animations into the page which either attract attention to a specific part of the page or which make the page easier to use
 - provide responses within the web page to various actions that your visitor takes
 - load new images, objects, or scripts into the web page without needing to reload the entire page
 - pass requests back to the server and handle responses from the server without the need for loading new pages
 - not everyone visiting your page will have JavaScript and so your page will still need to work for those who don't have JavaScript

sources:

- <https://en.wikipedia.org>
- https://wiki.haskell.org/Referential_transparency

APIs

Summary:

- `Math..exact`
 - for arithmetic overflow
- `Iterator`
- `Scanner`
- `String`
- `StringBuilderr`
- `Set (HashSet)`
- `List (Arraylist)`
- `LinkedList`
- `Stack`
- `Queue (LinkedList, PriorityQueue)`
- `PriorityQueue`
- `Map (HashMap)`

Iterator:

- `hasnext`
- `next`
- `remove`

Scanner:

- `findinline(Pattern)`
- `hasNext..int,byte,pattern()`
- `next...Int`

String:

- charAt
- codePointCount(int begin,int end) returns unicode(ASCII) points within range
- valueOf
- toLowerCase()

StringBuilder:

- append(..int,char,etc.
- charAt
- insert(int i, char,int, string...)
- toString

Set(HashSet):

- add
- clear
- clone
- contains
- isEmpty
- iterator
- remove
- size

List (ArrayList):

- add(E e),(int ind, E e)
- clear

APIs notes

- `get(int x)`
- `indexOf`
- `set(int ind, E e)`
- `remove`

LinkedList (LinkedList):

- `add(E e),(int i, E e)`
- `addFirst`
- `addLast`
- `get(int i)`
- `peek`
- `remove`
- `set`

Stack (Stack):

- `empty()`
- `peek`
- `pop`
- `push`
- `search(E e)`

Queue (LinkedList, PriorityQueue):

- Interface with multiple implementations

PriorityQueue (Binary Heap):

- `add(E e)`

APIs notes

- iterator
- poll()
- peek()

Map (HashMap):

- put
- size
- get
- contains

Comparator:

- implements `Comparator<T>`
- override compare method

Java Nuances

General Tips:

- Getter and setter
- Override and super
- Java automatically collects garbage
- &&/|| checks left first
- + strings makes a new string every time, if you want to do in a loop use stringBuilder(reduce memory)
- Everything in Java not explicitly set to something, is initialized to a zero value
 - references (anything that holds an object):null
 - int/short/byte:0
 - float/double:0.0
 - booleans: false.
 - array of something, all entries are also zeroed

Useful built in functions:

- Arrays
 - Arrays.binarySearch(arr, target)
 - * Negative value shows where it should be
 - Arrays.sort(arr)

Switch Statement:

- Example

```
switch (month) {  
    case 1: monthString = "January";  
        break;  
    case 2: monthString = "February";  
        break;  
    case 3: monthString = "March";  
        break;  
    case 4: monthString = "April";
```



```
        break;
    case 5: monthString = "May";
        break;
    case 6: monthString = "June";
        break;
    case 7: monthString = "July";
        break;
    case 8: monthString = "August";
        break;
    case 9: monthString = "September";
        break;
    case 10: monthString = "October";
        break;
    case 11: monthString = "November";
        break;
    case 12: monthString = "December";
        break;
    default: monthString = "Invalid month";
        break;
}
```

Breaking out of for loops:

- if you want to skip a particular iteration, use continue

```
for(int i=0 ; i<5 ; i++){
    if (i==2){
        continue;
    }
}
```

- if you want to break out of the immediate loop use break

```
for(int i=0 ; i<5 ; i++){
    if (i==2){
        break;
    }
}
```

- if there are 2 loop, outer and inner.... and you want to break out of both the loop from the inner loop, use break with label

```
lab1: for(int j=0 ; j<5 ; j++){
    for(int i=0 ; i<5 ; i++){
```

```
        if (i==2){  
            break lab1;  
        }  
    }  
}
```

Generics:

- Definition
 - generics are a facility of generic programming
 - * a style of computer programming in which algorithms are written in terms of types to-be-specified-later that are then instantiated when needed for specific types provided as parameters
 - ex: compiletime: List<String>, runtime:List
- Notes
 - in java, generics are only checked at compile time for type correctness
 - generic type information is then removed via a process called type erasure, to maintain compatibility with old JVM implementations, making it unavailable at runtime
- Sources
 - https://en.wikipedia.org/wiki/Generics_in_Java

Type Classifications:

- Concrete Types
 - concrete types describe object implementations, including memory layout and the code executed upon method invocation
 - the exact class of which an object is an instance not the more general set of the class and its subclasses
 - beware of falling into the trap of thinking that all concrete types are single classes!
 - Set of Exact Classes
 - ex: List x has concrete type ArrayList, LinkedList, ...
- Abstract Types
 - Abstract types, on the other hand, describe properties of objects
 - They do not distinguish between different implementations of the same behavior

- Java provides abstract types in the form of interfaces, which list the fields and operations that implementations must support

Access Modifiers:

- public
 - any class can access
 - accessible by entire application
- private
 - only accessible within the class
- protected
 - allow the class itself to access them
 - classes inside of the same package to access them
 - subclasses of that class to access them
- package protected
 - default
 - the same class and any class in the same package has access
 - protected minus the subclass unless subclass is in package
- Static: Belongs to class not an instance of the class

Things to override in new object (for hashing and equality uses):

- public int hashCode()
- public boolean equals(Object object)

ex: Tiger

```
@Override
public boolean equals(Object object) {
    boolean result = false;
    if (object == null || object.getClass() != getClass()) {
        result = false;
    } else {
        Tiger tiger = (Tiger) object;
        if (this.color == tiger.getColor()
            && this.stripePattern == tiger.getStripePattern()) {
            result = true;
        }
    }
}
```

```
    }  
  }  
  return result;  
}
```

Sources:

- <https://www.cs.utexas.edu/~scottm/cs307/codingSamples.htm>