Ad-Hoc polymorphism

Normal dynamic dispatch

Parametric Polymorphism

Type parameters

Inclusion Polymorphism

subtyping, etc.

Refactoring

1. **limiting access** to the client
2. **nesting classes:** move member classes into the parent class, and declare them static.
3. **merging classes**: merging a subclass with it's base class(for example, stackInt, push, and empty turns into stackInt and empty, and stackInt is no longer abstract(it has all the functionality of push)
4. **singleton pattern:** restricts instantiation of a class to one object
5. **using null as a normal value:** ex. having null represent push(empty(), 5);
6. **Inlining method calls:** replacing a call to a method with it's body (problem: you are substituting a method body for an expression)
7. **Relying on NullPointerExceptions**
8. **Precomputation**: i.e precomputing the size.

Programming Paradigms

Control Oriented
Data Oriented
Object Oriented
Abstract Oriented

- java is not truly object oriented. primitive types are not objects.

To Check is it is a BST:

Test **<u>ALL</u>** conditions

use Recursion - a simple recursive definition works best.

- For each node, check if max value in left subtree is smaller than the node and min value in right subtree greater than the node.
  - false if max of left is > current node
  - false in min of right is <= curr. node
  - false if left or right is not a bst
  - if the current node is empty, return true.

Amortized constant time

worst-case average time for a sequence of operations

if **x.equals(y)** then **x.hashCode() == y.hashCode()**

Reuse via adaptation

just using something

Reuse via inheritance

subclass is able to delegate to parent class

Reuse via composition

making a wrapper class that can delegate to/utilize the inner class

----

Adapter pattern

reusing by composition/wrapper class

Composite Pattern

class whose values refers to other classes, and whose methods manipulate those classes (like a node in a binary tree)

Caching

Storing some computation that may be used again - runs the computation when it's first needed, and then stores the value of that computation for further access.

Dynamic Types

<u>values</u> have types. each value has a notion of what type it is, but at compile-time, there is no notion of types.

Static Types

<u>expressions</u> have types. two things can have the same value, but different types.

Partition Types

like static types, but every expression has it's own unique type

----

A type constructor takes a type and returns another type. Java's Array constructor is an example of a type constructor.

Finding O(n) for a given function:

- If *f(x)* is a sum of several terms, the one with the largest growth rate is kept, and all others omitted.
- If *f(x)* is a product of several factors, any constants (terms in the product that do not depend on x) are omitted.

$f(x) \in O(g(x)) : |f(n)| \leq g(n) \cdot k$

f(x)'s growth is limited to g(x).

$f(x) \in \Omega(g(x)) : |f(n)| \geq g(n) \bullet k, k>0$

f(x)'s growth is bounded below by g

$f(x) \in \theta(g(x)) : g(n) \bullet k_1 \leq |f(n)| \leq g(n) \bullet k_2$

f(x)'s growth is bounded above and below by g asymptotically

Structural Types:

two types are compatible/equal if the structure of the type is the same.

```
type point = record
      a: double
      b: double
drocer

type pair = record
      a: double
      b: double
drocer
```

So A and B are equal.

Generative Types:

Every declaration of a type is different, so if you declare type pair once, and then declare it again, they are different types EVEN if they have the same structure or name.

Nominal Types:(JAVA has this.)

if two types have different names, those two types are different.

Private is visible only within the class
Default is that class and all subclasses
protected is in the same package
public is everywhere

StringBuilder and StringBuffer are the same, except stringBuffer is thread-safe.
sb.append(v) where it adds the string representation of v to the builder
reverse, delete(from, to), charAt(int i), sb.toString

Iterator.remove() will remove an element from the data representation, so something also accessing that data may assume the removed element still exists, throwing a ConcurrentModificationException.

Sum Types

if B and C are subclasses of A, then the type of A is a sum type of B and C.

Product Types

if A is a class with members of type B, C, and D, then the type of A is a product type of B, C, and D.

-largest possible int char is (2^31)-1
-char is 16 bits because the creators of java thought a unicode scalar would fit in 16 bits
-FDIV bug cost 475 million dollars
-**a type is a subtype of itself**
-Type parameters range over REFERENCE types
-**Covariance** of array: if B is a subtype of A, then B[] is a subtype of A[].

- **P** is the set of all problems that can be solved on a deterministic machine
- **NP** is the set of all problems for which any proposed solution can be verified in polynomial time

---

Optimization

Don't compute things that don't need to be computed
Don't recompute if you don't have to

precomputation
caching
memoization - after you compute something for a given input, store the returned value so you don't have to run the computation again.

dynamic programming - breaking it up into smaller subproblems.

divide and conquer - making the problem smaller in some way. Example: binary search tree searching

---

-Representing a finite map: search-

Algorithm	**worst**	**avg.**
linear assoc.	Ω(n)	θ(n)
search tree	θ(n)	θ(lg n)
hash table	θ(n)	θ(1)
R/B tree	O(lg n)	θ(lg n)