T204 Advanced OO Programming

Types

Java has 8 primitive types
○ boolean, byte, char, short, int, long, float and double

Java also has object types
○ String, BigInteger

Pimitive types are lowercase, while object types start with a capital letter.

java is a statically-typed language: types of all variables have to be known at compile time (before the program runs).

dynamically-typed languages like Python or Javascript, this kind of checking is deferred until runtime (while the program is running).

Checking

Three kinds of automatic checking that a language can provide:

1. ***Static checking***: the bug is found automatically before the program even runs.
2. ***Dynamic checking***: the bug is found automatically when the code is executed.
3. ***No checking***: the language doesn't help you find the error at all. You have to watch for it yourself, or end up with wrong answers!

dynamic checking will throw exception, but no checking only result error.

Static Checking

Static checking can catch:
● syntax errors, like extra punctuation or spurious words
● wrong names, e.g. Math.sine(2) (The right name is sin)
● wrong number of arguments, e.g. Math.sin(30, 20)
● wrong argument types, e.g. Math.sin("30")
● wrong return types, e.g. return "30"; from a function that's declared to return an int

Dynamic Checking

Dynamic checking can catch:
● illegal argument values, for example, the integer expression x/y is only erroneous when y is actually zero; otherwise it works!
so in this expression, divide-by-zero is not a static error, but a dynamic error
● unrepresentable return values, i.e. when the specific return value can't be represented in the type
● out-of-range indexes, e.g. using a negative or too-large index on a string
● calling a method on a null object reference

No checking

- Integer division, e.g. int a = 5 / 2;
- Integer overflow, e.g. byte a = 382387478324324;
- Special values in floating-point types, e.g. NaN

public and static

public means that any code, anywhere in your program, can refer to that method.
static means the method is associated with the class, not with an object.

Javadoc Comments and Documenting Assumptions

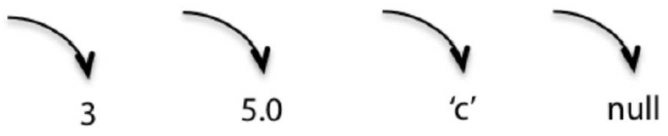
method signature: public static List hailstone(int n) , the parameters' types and return type.

```
/**
 * Compute a hailstone sequence.
 * For example, hailstone(5) = [5 16 8 4 2 1].
 * @param n starting number for sequence. Assumes n > 0.
 * @return hailstone sequence starting at n and ending with 1.
 */
```

Snapshot diagrams

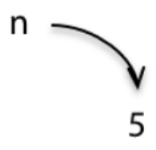
Snapshot diagrams represent the internal state of a program at runtime – its stack (methods in progress and their local variables) and its heap (objects that currently exist)

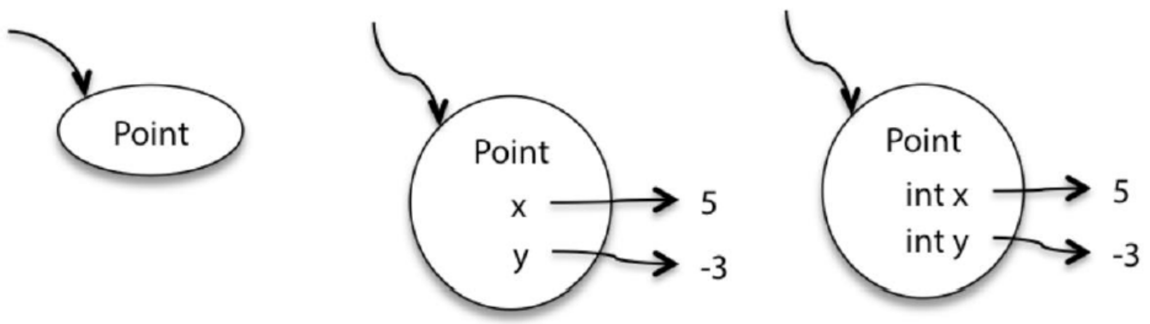
- Primitive values are represented by bare constants

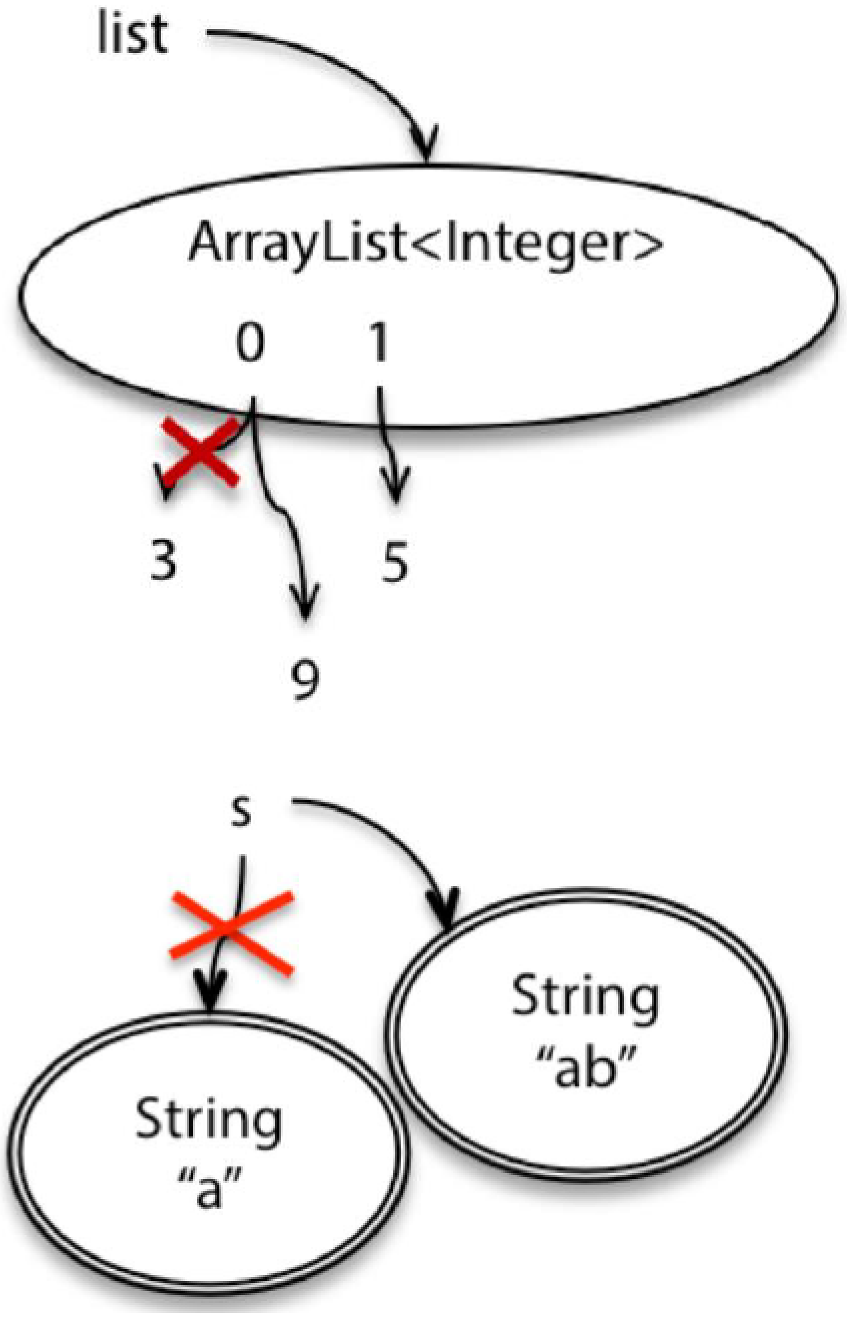


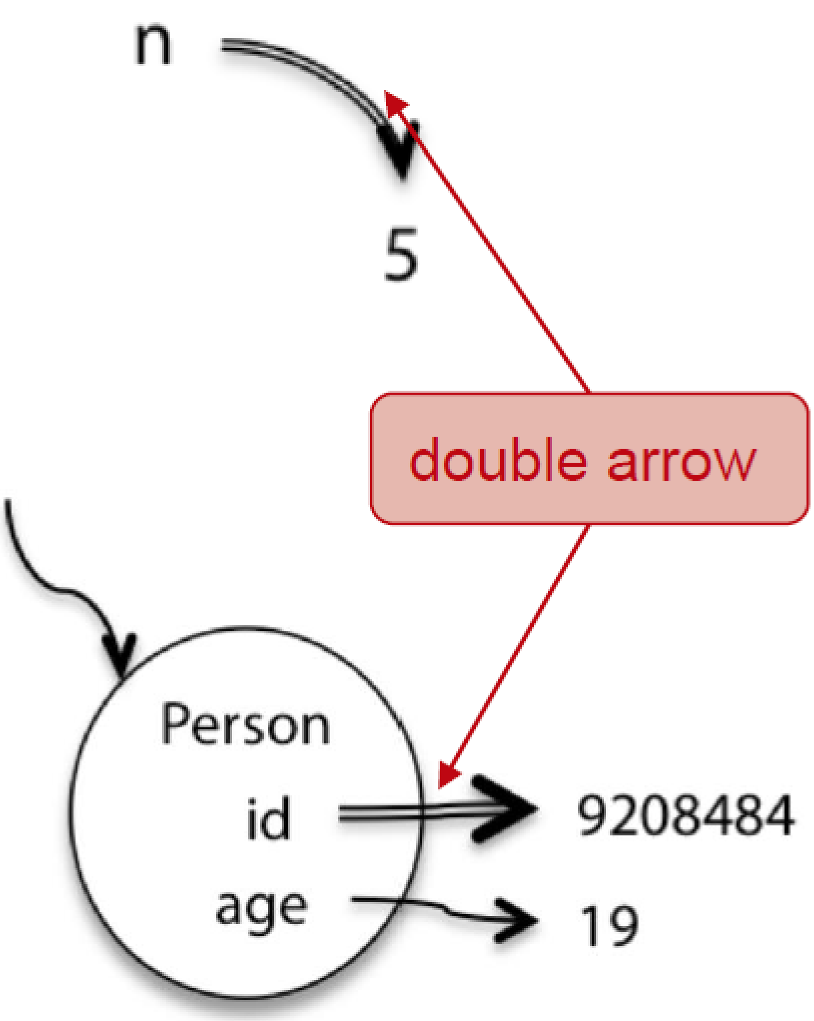
  - the incoming arrow is a reference to the value from a variable or an object field

- In its simplest form, a snapshot diagram shows a variable with an arrow pointing to its value






list
ArrayList<Integer>
0
1
3
5
9
s
String
"ab"
String
"a"



mutable and immutable

mutable value and immutable value

Immutable type: a type whose values can never change once they have been created, e.g. String.
mutable type: has methods that change the value of the object, e.g. StringBuilder.

mutable reference and immutable reference

mutable reference: variables that can point to other value, e.g. String a = "sd", a = "a";
immutable reference: variables cannot point to other value, e.g. variables declare with the keyword "final"

if a final variable points to a mutable object, then the variable cannot be reassigned, but the object it points to can still be mutated, e.g. final List a = {1, 2}, a = {2, 3};

Final

final can be used on both **parameters** and **local variables**

if a final variable points to a mutable object, then the variable cannot be reassigned , but the object it points to can still be mutated.

Map

A map stores key/value pairs , where each key has an associated value.

Interface

interfaces: define how these respective types work, but don’t provide implementation code.

- Recall our implementation of hailstone using array

```java
int[] a = new int[100];
int i = 0;
int n = 5;
while (n != 1) {
    a[i] = n;
    i++;
    if (n % 2 == 0) {
        n = n / 2;
    }
    else {
        n = 3 * n + 1;
    }
}
a[i] = n;
```

What would happen if we tried an n that turned out to have a very long hailstone sequence?
It wouldn't fit in a length-100 array
We have a bug !
Would Java catch the bug :

a. statically
b. dynamically
c. not at all ?