Programming with Data Structures

CMPSCI 187

Fall 2018

- **Please find a seat**
  - **Try to sit close to the center (the room will be pretty full!)**
- **Turn off or silence your mobile phone**
- **Turn off your other internet-enabled devices**

Reminders

- Read course webpage.
- Make sure you can log in to Piazza and Gradescope.
- Get iClicker **2** and register it in Moodle.
- Work on Project 1 (due by 3:59pm this Friday).
- Attend the first lab on Monday.
  - Bring your own laptop to the lab.

Lecture 2: Java Review

- **Java Primitive Data Types and Operations**
- **Objects, References, Aliasing, GC**
- **Parameter passing**
- Inheritance and dynamic typing
- Variable scope
- Conditional statements and loops
- Arrays
- Exceptions
- Programming Exercises

Java Programming Language

- Java is a high-level programming language. It lets us program with minimal knowledge of machine details.
  - Java programs are compiled into **.class** files
  - When you run the program, an interpreter (written in a lower-level language) executes the class file.

- How is this different from languages like C/C++?
- How is this different from languages like Python, Javascript?

Primitive Data Types

- All data in Java eventually reduces to primitive data types. Examples:
  - Integral:
```
int i = 100;
long x = 1234567890L;
```
  - Decimal:
```
float height = 5.9f;
double weight = 160.5;
```
  - Logical: `boolean hasName = true;`
  - Character: `char answer = 'y';`

Primitive Data Types

- Type casts are generally done automatically from a lower precision type to higher precision type. **Otherwise, you must use explicit type casts.**
- Each primitive type has a corresponding **wrapper class** to allow them to be used as objects.

Examples:

- `Integer`
- `Float`
- `Character`
- `Boolean`

The purpose of these wrapper classes will be clear when we learn about generics.

Primitive Data Types

- There are lots of operations on these types:
  - Arithmetic: `+` `-` `*` `/` `%` `++` `--`
  - Bitwise: `&` `|` `^` `~` `>>` `<<` `>>>`
  - Relational: `==` `!=` `>=` `<=` `<` `>`
  - Logical: `&&` `||`
  - Assignment: `=` `+=` `-=` `*=` `/=` …
  - **Ternary (conditional):** `?:`

Clicker question #1

```
int x = 1;
int y = 5;
System.out.println(x / y);
```

What is the output?

a. 3
b. 2.5
c. 0
d. 0.2
e. 2

Clicker question #2

```
int x = 2;
int y = 5;
double d = x / y;
System.out.println(d);
```

What is the output?

a. 0
b. 2
c. 0.4
d. 0.0
e. 2.5

Explicit type casts

```
int x = 2;
int y = 5;
double d = (double)x / y;
System.out.println(d);
```

What is the output?

a. 2

b. 0.4

c. 2.5

d. 0

e. 0.0

Primitive Data Types

- **Ternary (conditional): `? :`**
  `(condition)?(statement 1):(statement 2)`

- **Example:**

```
int z = (x > y) ? x : y;
return (object==null) ? 0 : object.value;
System.out.print((x%2==1)?"Odd":"Even");
```

- **Nested:**

```
m=(a>b)?((a>c)?a:c):((b>c)?b:c);
```

- **Demo**

Objects and References

- An **object** is a bundle of data and behavior. In Java: a set of variables and associated methods.
- Defined by classes. Example:

```
class Apple {
  private float weight, size;
  public float getWeight() { return weight;}
  public void setWeight(float w)
    { weight=w; }
}
```

- What's the difference between class definition and an instance of the class?

Objects and References

- When an object (instance) is created (using the **`new`** keyword), memory is allocated for the object.

```
Apple apple = new Apple();
```

- The memory address is called a “pointer” or “reference” to the object.
- Variable `apple` holds the reference (i.e. the memory address) to the newly created Apple object.
- In Java, you have references only to objects, not to primitive data types (different from C/C++)

Objects and References

- **Assigning** a reference to another variable does **NOT** create another instance — it merely copies the pointer from the first variable to the second:

```
String message = new String(“Hi!”);

String hi = message;
```

- Thus both variables reference the same object, and we say variables hi and message are ‘**alias**’ of each other.

Variables and values

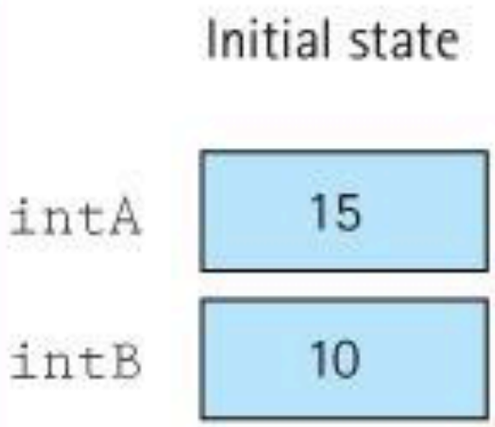
Differently-named variables are independent.

For primitive types, this is straightforward:

```
int x = 1;
int y = 99;
y = x;
System.out.println(y); // 1
y = 100;
System.out.println(x); // 1
System.out.println(y); // 100
```

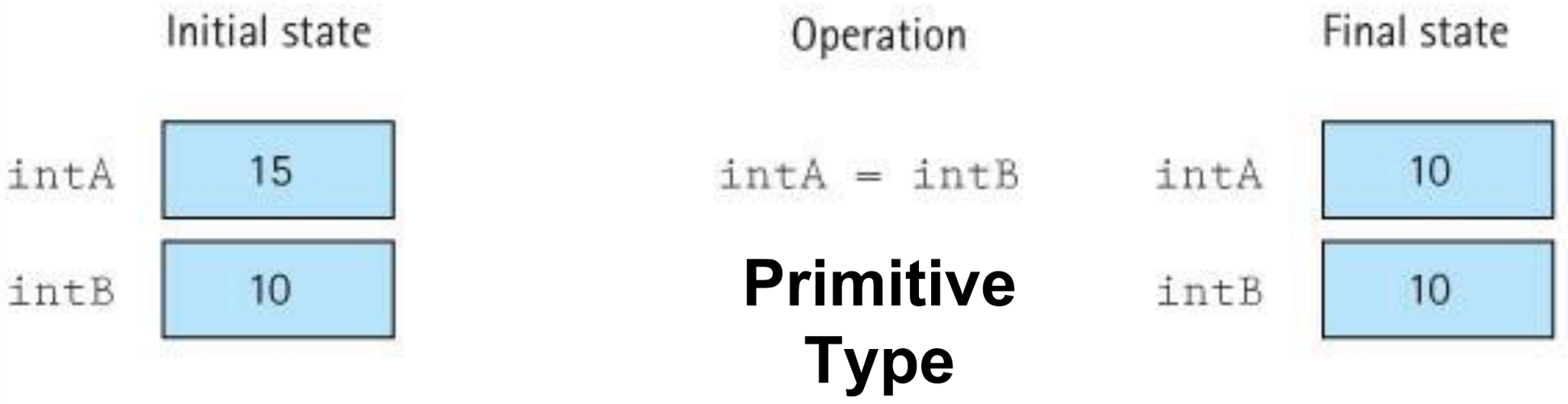
For objects, it’s the same, but you have to be careful...

Assignments

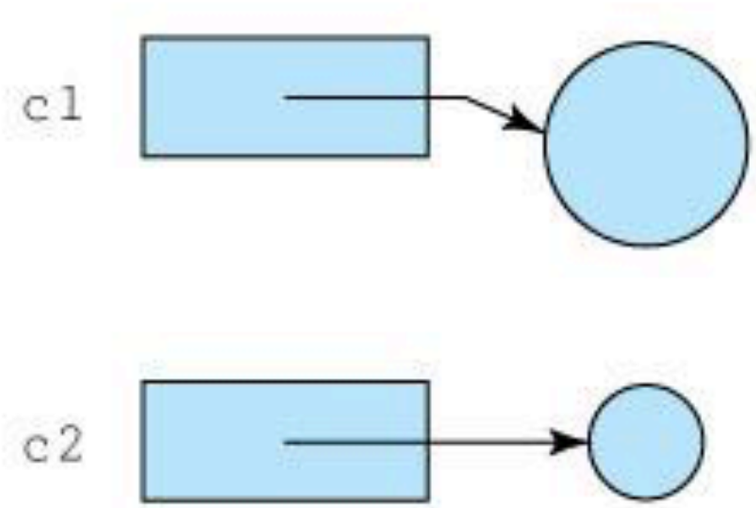


Primitive Type

Assignments



Primitive Type



Object Type (e.g. Circle class)

Aliasing

An object variable actually contains a reference (i.e. pointer) to the object.

- Multiple variables can point to the same object!

```
Person x = new Person("Homer");
Person y = new Person("Marge");
y = x;
System.out.println(y.getName()); // Homer
y.setName("Bart");
System.out.println(x.getName()); // Bart
System.out.println(y.getName()); // Bart
```

Aliasing

```
Person x = new Person("Homer");
Person y = new Person("Marge");
y = x;
System.out.println(y.getName()); // Homer
y.setName("Bart");
System.out.println(x.getName()); // Bart
System.out.println(y.getName()); // Bart
```

This starts to become especially confusing when you use mutators (i.e. set methods).

Aliasing is a notorious problem for beginners!

When in doubt, draw pictures to help you figure it out!

Clicker question #3

```
Dog cardie = new Dog();
Dog duncan = cardie;
cardie.setAge(5);
duncan.setAge(7);
int sum = cardie.getAge() + duncan.getAge();
```

- The value of sum at the end is:

(a) 10 (b) 8

(c) 14 (d) 12

Comparing two objects using ==

- When using the == operator to compare two **objects** A and B, the result is true only if they reference the same object (i.e. contain the same pointer), regardless of whether the data members in A and B are equal or not.
- Unlike C++, Java does not allow operator overloading. Hence if you want to compare the **content** in objects A and B, you need to define a custom method.
  - Example: String's **`.equals()`** method.

Clicker question #4

```
Integer a = new Integer(187);
Integer b = new Integer(187);
System.out.println(a==b);
```

- The output is

  (a) 187 (b) null

  (c) false (d) true

Avoid Aliasing

If you want to avoid aliasing, you should explicitly clone an existing object to a new object (typically done through a **copy constructor**)

```
Person x = new Person("Homer");
Person y = new Person("Marge");
y = new Person(x);
System.out.println(y.getName()); // Homer
y.setName("Bart");
System.out.println(x.getName()); // Homer
System.out.println(y.getName()); // Bart
```

Garbage Collector (GC)

- A method for automatic memory management. Specifically, GC automatically reclaims memory occupied by objects that are no longer in use.
- Different from C/C++ which requires explicit memory management (manually deleting unused memory)
- If an object is referenced by some variable, it's considered 'in use' and the GC will keep it.
- If it's no longer referenced by any variable, it's marked 'garbage' and will be reclaimed by GC at some point.

Garbage Collector (GC)

- **Advantages**:
  - No more dangling pointers
  - Reduces memory leaks (i.e. memory that's no longer needed but which the program fails to free, resulting in memory exhaustion).
- **Disadvantages**:
  - Potential performance impact
  - Requires additional resources to keep track of memory to be automatically reclaimed.

Static Variables and Methods

- Some variables and methods are declared as **static**. Examples:

```
class Apple {
  public static int id;
  public float weight;
  public static int getID();
}
```

- How are these different from other (non-static) variables and methods?

Static Variables and Methods

- Static variables exist (are allocated in memory) without any class instantiation.
  - Think of them as '**globally existing**' variables.
  - In contrast, non-static variables are only allocated when you create / instantiate a new object.
- Objects of the same class refer to the same static variables (one global copy for all objects).
  - In contrast, non-static variables have unique local copies in each different object.
- Example: `Math.PI;`

Static Variables and Methods

- Static variables/methods (if public) may be called directly using the class name. Example:

```
System.out.println(Apple.id);
Apple.getID();
Math.random();
```

  - In contrast, non-static variables/methods cannot be called without a class instance (object).
- Static methods do not have access to ‘this’ pointer, hence cannot call non-static methods or use non-static variables.
  - What about the reverse?

What is wrong here?

```
class Two {
  public static void main(String[] args) {
    int x = 0;
    System.out.println("x = " + x);
    x = fxn(x);
    System.out.println("x = " + x);
  }
  int fxn(int y) {
    y = 5;
    return y;
  }
}
```