Memory

Chris Piech
CS106A, Stanford University
Learning Goals

1. Be able to trace memory with references
Write this program
Who thinks this prints `true`?

```java
public void run() {
    GRect first = new GRect(20, 30);
    GRect second = new GRect(20, 30);
    println(first == second);
}
```
Who thinks this prints `true`?

```java
public void run() {
    int x = 5;
    int y = 5;
    println(x == y);
}
```
Who thinks this prints `true`?

```java
private GRect first = new GRect(20, 30);
public void run() {
    first.setFilled(true);
    add(first, 0, 0);
    GObject second = getElementAt(1, 1);
    println(first == second);
}
```
Class of the 10 keys
Advanced memory model
Core memory model
public void run() {
    println(toInches(5));
}

private int toInches(int feet){
    int result = feet * 12;
    return result;
}
public void run() {
    println(toInches(5));
}

private int toInches(int feet) {
    int result = feet * 12;
    return result;
}
public void run() {
    println(toInches(5));
}

private int toInches(int feet) {
    int result = feet * 12;
    return result;
}
```java
public void run() {
    println(toInches(5));
}

private int toInches(int feet) {
    int result = feet * 12;
    return result;
}
```

Stack Diagrams

**run**

**toInches**

- **feet**
  - 5
- **result**
  - 60
public void run() {
    println(toInches(5));
}

private int toInches(int feet) {
    int result = feet * 12;
    return result;
}
public void run() {
    println(toInches(5));
}

private int toInches(int feet) {
    int result = feet * 12;
    return result;
}
Aside: Actual Memory
What is a bucket

feet

5
What is a bucket

* Each bucket or “word” holds 64 bits

** don’t think on the binary level (yet)
#0: variables have fixed size buckets to store values
End aside
# Primitives vs Classes

<table>
<thead>
<tr>
<th>Primitive Variable Types</th>
<th>Class Variable Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>GRect</td>
</tr>
<tr>
<td>double</td>
<td>GOval</td>
</tr>
<tr>
<td>char</td>
<td>Gline</td>
</tr>
<tr>
<td>boolean</td>
<td>Color</td>
</tr>
</tbody>
</table>

Class variables (aka objects)
1. Have upper camel case types
2. You can call methods on them
3. Are constructed using `new`
4. Are stored in a special way
Primitives vs Classes

Primitive Variable Types
- int
- double
- char
- boolean

Class Variable Types
- GRect
- GOval
- Gline
- Color

Class variables (aka objects)
1. Have upper camel case types
2. You can call methods on them
3. Are constructed using `new`
4. Are stored in a special way
How do you share Wikipedia articles?

Antelope Canyon Article

Antelope Canyon is a slot canyon in the American Southwest. It is located on Navajo land east of Page, Arizona. Antelope Canyon includes two separate, photogenic slot canyon sections, referred to individually as Upper Antelope Canyon or The Crack; and Antelope Canyon or The Corkscrew.[2]

The Navajo name for Upper Antelope Canyon is Tsé bighánílíní, which means "the place where water runs through rocks." Lower Antelope Canyon is Hazéistází (advertised as "Hazéistwazi" by the Navajo Parks and Recreation Department), or "spiral rock arches." Both are located within the LeChee Chapter of the Navajo Nation.[4]

https://en.wikipedia.org/wiki/Antelope_Canyon
All of todays class:

Objects store addresses
(which are like URLs)
What does an object store?
Objects store addresses
(which are like URLs)
A Variable love story

By Chris Piech
A Variable story

origin

By Chris Piech

Nick Troccoli
Once upon a time...
...a variable x was born!

int x;
...a variable x was born!

```c
int x;
```
x was a primitive variable...

```plaintext
int x;
```

Aww...!

It’s so cuuuute!
...and its parents loved it very much.

```c
int x;
```

We should give it.... value 27!
...and its parents loved it very much.

\[ x = 27; \]

We should give it... value 27!
A few years later, the parents decided to have another variable.
...and a variable rect was born!

GRect rect;
rect was an object variable...

GRect rect;

Who’s a cute GRect???

It’s so square!
...and its parents loved it very much.

GRect rect;

We should make it.... a big, strong GRect!
...and its parents loved it very much.

```java
GRect rect = new GRect(0, 0, 50, 50);
```

We should make it... a big, strong GRect!
...but rect’s box was not big enough for an object!

GRect rect = new GRect(0, 0, 50, 50);

That box isn’t big enough to store everything about a GRect!
GRect rect = new GRect(0, 0, 50, 50);

x = 0, y = 0
width = 50
height = 50
...
In practice
public void run() {
    GRect r = null;
}

Method memory                                      Object memory
public void run() {
    GRect r = null;
}

Method memory

run

r null

Object memory
Wahoo!
public void run() {
    GRect r = new GRect(50, 50);
}

Method memory

Object memory
public void run() {
    GRect r = new GRect(50, 50);
}

Method memory

Object memory

memory.com/18
public void run() {
    GRect r = new GRect(50, 50);
}

Method memory

Object memory

run

r memory.com/18

memory.com/18
public void run() {
    GRect r = new GRect(50, 50);
}

Method memory

Object memory
public void run() {
    GRect r = new GRect(50, 50);
}

Method memory

run

r

Object memory
```java
public void run() {
    GRect r = new GRect(50, 50);
    r.setColor(Color.BLUE);
    r.setFilled(true);
}
```
public void run() {
    GRect r = new GRect(50, 50);
    r.setColor(Color.BLUE);
    r.setFilled(true);
}

Method memory

Object memory

run

r
public void run() {
    GRect r = new GRect(50, 50);
    r.setColor(Color.BLUE);
    r.setFilled(true);
}

Method memory

Object memory

run

r
#1: **new** allocates memory for objects

* The data for an object can’t always fit inside a fixed size bucket
#2: object variables store addresses

#ultimatekey
```java
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}
```

stack

heap

run
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}

---

Stack:

```
run
```

Heap:
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}

stack

heap

run
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}

stack

heap

run
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}

stack

heap 42

run

Piech, CS106A, Stanford University
```java
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}
```
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}

run

heap 42

stack

run

img 42
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}

stack

heap

run

img
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}
#3: GImages look impressive but don’t take much extra work
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}

stack

heap

run

first

second
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}

stack

heap

32
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}

stack

heap

run

first

second

32

32
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}

stack

run

first

second

heap

32
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}

stack

run

first 32

second 32

heap

32
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLU)
    add(first, 0, 0);
}
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}

---

[Diagram showing stack and heap with variables and values]
```java
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
    add(second, 10, 10);
}
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
    add(second, 10, 10);
}

stack

run

first

second

heap

32

32

32
#4: when you use the = operator with objects, it copies the address
What does an object store?
Objects store addresses (which are like URLs)
Passing by “Reference”
Primitives pass by value

// NOTE: This program is buggy!!

public void run() {  
    int x = 3;  
    addFive(x);  
    println("x = " + x);  
}

private void addFive(int x) {  
    x += 5;  
}

* This is probably the single more important example to understand in CS106A
Objects pass by reference

// NOTE: This program is awesome!!

public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}

* This is probably the single more important example to understand in CS106A
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUe);
    object.setFilled(true);
}
```java
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```java
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```
```java
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFill(true);
}
```java
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
#5: when you pass (or return) an Object, the address is passed.

Aka reference
What does an object store?
Objects store addresses (which are like URLs)
GRect paddle = new GRect(20, 30);
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```java
GRect paddle = new GRect(20, 30);
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```
GRect paddle = new GRect(20, 30);
  public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
  }

Instance Variables

paddle

run

heap

18
GRect paddle = new GRect(20, 30);
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}

Instance Variables

paddle

18

run
#7: there is space for all instance variables. They are accessible by the entire class
#8: instance variables are initialized before run is called
Question: what does this program do?

```
GRect paddle = new GRect(getWidth(), getHeight());
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```

Answer: makes a square that is 0 by 0 since `getWidth` is called before the screen has been made.
#9: for objects == checks if the variables store the same address
Recall the start of class?
Who thinks this prints true?

```java
public void run() {
    GRect first = new GRect(20, 30);
    GRect second = new GRect(20, 30);
    println(first == second);
}
```
Who thinks this prints \textbf{true}?

```java
public void run() {
    int x = 5;
    int y = 5;
    println(x == y);
}
```
Who thinks this prints `true`?

```java
private GRect first = new GRect(20, 30);
public void run() {
    first.setFilled(true);
    add(first, 0, 0);
    GObject second = getElementAt(1, 1);
    println(first == second);
}
```
What does an object store?
Objects store addresses
(which are like URLs)
Milestones

Milestone 1

Milestone 2
Finish Up
Learning Goals

1. Be able to trace memory with references