#### Topic 3 References and Object Variables "Thou shalt not follow the NULL pointer, for chaos and madness await thee at its end."

- <u>Henry Spencer</u>



### **Object Variables**

- object variables are declared by stating the class name / data type and then the variable name
  - same as primitives
  - in Java there are hundreds of built in classes.
    - show the API page
  - don't learn the classes, learn how to read and use a class interface (the users manual)
- objects are *complex* variables.
  - They have an internal state and various behaviors that can either change the state or simply tell something about the object



- So now there are 2 Rectangle objects right?
- Not so much.
- Object variables in Java are actually references to objects, not the objects themselves!
  - object variables store the memory address of an object of the proper type *not* an object of the proper type.
  - contrast this with primitive variables

### The Pointer Sidetrack

- IMPORTANT!! <u>This material may</u> <u>seem a bit abstract, but it is often the</u> <u>cause of many a programmers logic error</u>
- A pointer is a variable that stores the memory address of where another variable is stored



- In some languages you can have bound variables and dynamic variables of any type
  - a bound variable is one that is associated with a particular portion of memory that cannot be changed
- Example C++, can have an integer variable or a integer pointer (which is still a variable) int intVar; // a int var int \* intPtr; //pointer to an int var

#### Pointer Variables in C++





#### **Pointer Complications**

C++ allows actual variables and pointers to variables of any type. Things get complicated and confusing very quickly

int intVar = 5; // a int var int \* intPtr; //pointer to an int var intPtr = new int; // allocate memory \*intPtr = 12; /\* assign the integer being pointed to the value of 12. Must dereference the pointer. i.e. get to the thing being pointed at\*/ cout << intPtr << "\t" << \*intPtr << "\t" << &intPtr << endl; // 3 different ways of manipulating intPtr

#### In C++ you can work directly with the memory address stored in intPtr

- increment it, assign it other memory addresses, pointer "arithmetic"

CS 307 Fundamentals of Computer Science

Given the following C++ declarations how would the variable intPtr be made to refer to the variable intVar?

```
intVar = 5;
intPtr = new int;
```

- A.intPtr = intVar;
- B.intPtr = \*intVar;
- C. \*intPtr = intVar;
- D.intPtr = &intVar;
- E.intPtr = intVar;

And Now for Something Completely Different...

#### Thanks Nick…

#### Link to Bink

#### Pointer Fun with Binky



by Nick Parlante This is document 104 in the Stanford CS Education Library — please see cslibrary.stanford.edu for this video, its associated documents, and other free educational materials.

Copyright © 1999 Nick Parlante. See copyright panel for redistribution terms. Carpe Post Meridiem!

CS 307 Fundamentals of Computer Science

# **Benefit of Pointers**

- Why have pointers?
- To allow the sharing of a variable
  - If several variables(objects, records, structs) need access to another single variable two alternatives
    - 1. keep multiple copies of variable.

2. share the data with each variable keeping a reference to the needed data



#### **Time Space Trade Off**

 Often the case that algorithms / solutions an be made faster by using more space (memory) or can use less space at the expense of being slower.



#### time to complete

CS 307 Fundamentals of Computer Science

#### **More Benefits**

- Allow dynamic allocation of memory
  - get it only when needed (*stack* memory and *heap* memory)
- Allow linked data structures such as linked lists and binary trees
  - incredibly useful for certain types of problems
- Pointers are in fact necessary in a language like Java where *polymorphism* is so prevalent (more on this later)
- Now the good news
  - In Java most of the complications and difficulties inherent with dealing with pointers are removed by some simplifications in the language

#### **Dynamic Memory Allocation**

Your program has two chunks of memory to work with: *Stack memory* (or the *runtime Stack*) and *Heap memory* 

When a Java program starts it receives two chunks of memory one for the Stack and one for the Heap.

Things that use Stack memory: local variables, parameters, and information about methods that are in progress.

Things that use Heap memory: everything that is allocated using the new operator.



```
void toyCodeForMemory(int x)
{ int y = 10;
    x += y;
    String s = new String("Hello");
    System.out.println(x + " " + y + s);
```

CS 307 Fundamentals of Computer Science

#### How Much Memory?

#### How big is the Heap?

System.out.println("Heap size is " +
Runtime.getRuntime().totalMemory());

# How much of the Heap is available? System.out.println("Available memory: " Runtime.getRuntime().freeMemory());

#### **References in Java**

- In Java all primitive variables are value variables. (real, actual, direct?)
  - it is impossible to have an integer pointer or a pointer to any variable of one of the primitive data types
- All object variables are actually reference variables (essentially store a memory address) to objects.
  - it is impossible to have anything but references to objects. You can never have a plain object variable

CS 307 Fundamentals of Computer Science

### Back to the Rectangle Objects

- rect1 and rect2 are variables that store the memory addresses of Rectangle objects
- right now they are uninitialized and since they are local, variables may not be used until they are given some value

null is used to indicate an object variable is not pointing / naming / referring to any Rectangle object.

# **Creating Objects**

- Declaring object variables does not create objects.
  - It merely sets aside space to hold the memory address of an object.
  - The object must be created by using the new operator and calling a *constructor* for that object

- For all objects, the memory needed to store the objects, is allocated dynamically using the new operator and a constructor call. (Strings are a special case.)
  - constructors are similar to methods, but they are used to initialize objects

CS 307 Fundamentals of References and Object Variables Computer Science

#### The Yellow Sticky Analogy



CS 307 Fundamentals of Computer Science

# Pointers in Java

- Is this easier?
  - primitives one thing, objects another?
- can't get at the memory address the pointer stores as in C++
  - although try this:

```
Object obj = new Object();
```

```
System.out.println( obj.toString() );
```

- dereferencing occurs automatically
- because of the consistency the distinction between an object and an object reference can be blurred
  - "pass an object to the method" versus "pass an object reference to the method
- Need to be clear when dealing with memory address of object and when dealing with the object itself

## Working with Objects

 Once an object is created and an object variable points to it then Object may be manipulated via its methods

```
Rectangle r1 = new Rectangle();
r1.resize(100, 200);
r1.setLocation(10, 20);
int area = r1.getWidth() * r1.getHeight();
Rectangle r2 = null;
r2.resize( r1.getWidth(), r1.getHeight() * 2 );
// uh-oh!
```

- Use the dot operator to deference an object variable and *invoke* one of the objects behaviors
- Available behaviors are spelled out in the class of the object, (the data type of the object)

CS 307 Fundamentals of Computer Science

#### What's the Output?

```
public void objectVariables()
       Rectangle rect1 = new Rectangle(5, 10, 15, 20);
       Rectangle rect2 = new Rectangle(5, 10, 15, 20);;
       System.out.println("rect 1: " + rect1.toString() );
       System.out.println("rect 2: " + rect2.toString() );
       // Line 1
       System.out.println("rect1 == rect2: " + (rect1 == rect2));
       rect1 = rect2;
       rect2.setSize(50, 100); // (newWidth, newHeight)
       // Line 2
       System.out.println("rect 1: " + rect1.toString() );
       System.out.println("rect 2: " + rect2.toString() );
       System.out.println("rect1 == rect2: " + (rect1 == rect2));
       int x = 12;
       int y = 12;
       // Line 3
       System.out.println("x == y: " + (x == y));
       x = 5;
       v = x;
       x = 10;
       System.out.println("x == y: " + (x == y));
       // Line 4
       System.out.println("x value: " + x + ", y value: " + y);
```

What is output by the line of code marked Line 1?

- A.rect1 == rect2: true
- **B.**rect1 == rect2: rect1 == rect2
- **C**.rect1 == rect2: false
- D.intPtr = &intVar;
- **E**.rect1 == rect2: 0

What will be the width and height of the Rectangle object rect1 refers to at the line of code marked Line 2?

- A. width = 15, height = 20
- B. width = 20, height = 15
- C. width = -1, height = -1
- D. width = 0, height = 0
- E. width = 50, height = 100

What is output by the line of code marked Line 3?

A.x == y: 0
B.x == y: 1
C.x == y: true
D.x == y: x == y
E.x == y: false

What is output by the line of code marked Line 4?

A.x value: 5, y value: 5
B.x value: 10, y value: 5
C.x value: 0, y value: 0
D.x value: 5, y value: 10;
E.x value: 10, y value: 10

#### **Equality versus Identity**

A man walks into a pizza parlor, sits down, and tells the waiter, "I'll have what that lady over there is eating." The waiter walks over to the indicated lady, picks up the pizza that is resting in front of her, and sets it back down in from of the man's table.

- confusion over equality and identity
- Identity: two things are in fact the same thing
- equality: two things are for all practical purposes alike, but not the exact same thing
- == versus the .equals method
  - use the .equals method when you want to check the contents of the pointee, use == when you want to check memory addresses

CS 307 Fundamentals of Computer Science

#### Just Like the Real World

- Objects variables are merely names for objects
- Objects may have multiple names
  - meaning there are multiple object variables referring to the same object (sharing)



### The Garbage Collector

Rectangle rect1 = new Rectangle(2,4,10,10);
Rectangle rect2 = new Rectangle(5,10,20,30);
// (x, y, width, height)
rect1 = rect2;
/\* what happened to the Rectangle Object
 rect1 was pointing at?
\*/

If objects are allocated dynamically with new how are they deallocated?

- delete in C++

If an object becomes isolated (no longer is in scope), that is has no references to it, it is garbage and the Java Virtual Machine garbage collector will reclaim this memory AUTOMATICALLY!

#### **Objects as Parameters**

- All parameters in Java are value parameters
- The method receives a copy of the parameter, not the actual variable passed
- Makes it impossible to change a primitive parameter
- implications for objects? (which are references)
  - behavior that is similar to a reference parameter, with a few minor, but crucial differences
  - "Reference parameter like behavior for the pointee."

#### Immutable Objects

- Some classes create *immutable* objects
- Once created these objects cannot be changed
  - note the difference between objects and object variables
- Most immediate example is the String class
- String objects are immutable
- Why might this be useful?

```
String name = "Mike";
String sameName = name;
name += " " + "David" + " " + "Scott";
System.out.println( name );
System.out.println( sameName );
```