Lec 7

Classes, Memory
Classes
Classes

• Constructor
• State
• Behaviors
public class Point{

   private double x;
   private double y;

   public Point(double x_in, double y_in){
      this.x = x_in;
      this.y = y_in;
   }

   public static void main( String[] args ){
      double d = 42.0;
      Point p;
      p = new Point(d, 42.0);

      Point q = new Point(p.x, 42)
   }
}
class Phone{
    ArrayList<Contacts> contacts;

    public Phone() {
        contacts = new ArrayList<>();
    }

    public void addContact(Contact c) {
        contacts.add(c);
    }

    public void removeContact(Contact c) {
        contacts.remove(c);
    }

    public void call(Contact c) {
        //do magic
    }
}
Phone

contacts: ArrayList<Contacts>

Phone()
addContact(c: Contact): void
removeContact(c: Contact): void
call(c: Contact): void
public vs. private vs. protected

class Phone{
    ArrayList<Contacts> contacts;

    public Phone() {
        contacts = new ArrayList<>();
    }

    public void addContact(Contact c) {
        contacts.add(c);
    }

    public void removeContact(Contact c) {
        contacts.remove(c);
    }

    public void call(Contact c) {
        //do magic
    }
}

Which of these statements does not exist in TV

- channel: int
- on: boolean

+TV()
+turnOn(): void
+turnOff(): void
+setChannel(newChannel: int): void

a) private int channel;
b) public void turnOn() {}
c) public boolean on;
d) private int channel() {}
e) (c) and (d)
Constructors

• Same name as the class, no return type
  
  ```java
  public Chalk(String myColor) {
      color = myColor;
  }
  ```

• Called automatically by `new` operator

• Often overloaded
  
  – Constructor with no parameters is called the default constructor
What gets printed

public class Chalk {
    String color;
    public Chalk() {
        color = "black";
    }
    public Chalk(String newColor) {
        color = newColor;
    }
    void write(String word) {
        System.out.println("/" + color + "/ " + word);
    }
}

public class Example{
    public static void main(String[] a) {
        Chalk c1 = new Chalk();
        Chalk c2 = new Chalk("green");

        c1.write("grass");
        c2.write("dress");
    }
}

A) /black/ grass, /black/ dress
B) /black/ grass, /green/ dress
C) /green/ grass, /green/ dress
D) /black/ dress, /black/ grass
E) None of the above
this
• Refers to “this instance” of an object
  – Think of as this = object’s memory address
• Used for:
  1. Disambiguate variable references
  2. Call alternate constructors from within a constructor
  3. Pass the current class instance as an argument to a method of another object
public class Chalk {
    private String color;
    
    public Chalk(String color) {
        this.color = color;
    }
}
public class Chalk {
    private String color;

    public Chalk() {
        this("white");
    }

    public Chalk(String color) {
        this.color = color;
    }
}
public class FBUser{
    private ArrayList<FBUser> friends;

    public FBUser(){
        friends = new ArrayList<>();
    }

    public void acceptFriend(FBUser newFriend) { 
        addFriend(newFriend);
        newFriend.addFriend(this);
    }

    public void addFriend(FBUser newFriend) {
        friends.add(friend);
    }
}
public class ChalkBox {
    private Chalk piece;
    public ChalkBox() {
        piece = new Chalk("green", this);
    }
    public void useChalk(String word) {
        piece.write(word);
        return;
    }
}

public class Chalk {
    private ChalkBox box;
    private String color;

    public Chalk(ChalkBox box) {
        this("white", box);
    }
    public Chalk(String color, ChalkBox box) {
        this.color = color;
        this.box = box;
    }
    public void write(String word) {
        System.out.println("/" + color + "/ " + word + " from box: " + box);
        return;
    }
}

public class Example {
    public static void main(…) {
        ChalkBox c = new ChalkBox();
        c.useChalk("hi");
    }
}
Memory
Memory

• Static: allocated once and last for the duration of a program
• Stack: region where a method's local variables are allocated during a method call
• Heap: region where the "new" operator allocates memory for objects
Stack

• Method call adds local vars and params to stack
• Return removes vars from stack
• Last In First Out (LIFO)
public class Ex {
    public static void main(…) {
        int i = 0;
        char c = ‘A’;
        foo(i);
    }
    public static void foo(int k) {
        double d = k * 2;
        bar();
        return;
    }
    public static void bar() {
        int z = 10;
        System.out.println(z);
        return;
    }
}
public class Ex {
    public static void main(…) {
        int[] array = new int[10];
        int value = 32;
        Object o = new Object();
    }
}
Heap

• Automatically managed by Java Garbage Collector
  – Keeps track of variables referring to object. Once it gets to 0, mark for deletion
**static**

- Allocated by compiler
- Address does not change
- static = “does not change”

- Instance does not have to be constructed before using static method/variable
  - Ex. main(): no instance was constructed
  - At run time “java MyClass” is actually running MyClass.main()
static methods

```java
public class Chalk {
    private String color;

    public Chalk(String color) {
        this.color = color;
    }

    public static void write(String word) {
        System.out.println(word);
    }
}

public class Example {
    public static void main(…) {
        Chalk.write("hi");
    }
}
```
static variables

• Common to all instances
  – One fixed memory location
public class Chalk {
    private static int numPieces = 0;
    private String color;

    public Chalk(String color) {
        this.numPieces++;
        this.color = color;
    }
    public static int getNumPieces() {
        return numPieces;
    }
}

public class Example {
    public static void main(…) {
        Chalk c1 = new Chalk("white");
        Chalk c2 = new Chalk("green");
        Chalk c3 = c1;
        int i = c3.getNumPieces();
        System.out.println(i);
    }
}

A) Compiler error
B) 0
C) 1
D) 2
E) 3