CS 106-A

Final

Tom McDonough

**Problem 1:**

When passing an object into a method you are actually passing a reference to that object; when you pass a primitive type (such as an int), you are passing actual data (in this case, a copy of the variable that you passed in). In the case of a reference, you tell the method to make changes to the object that you referred it to, which lives outside of the method. When you pass the method a primitive, it actually makes a copy of the original variable that was passed in and does its work on a new variable, so the original variable remains intact.

Sets an integer tmp to value 50

Cycles through the array, starting at index 1 and going up to index 4

The first iteration sets the value at index 1 equal to the value at index 0, which is 10.

The next 3 iterations all end up setting that index to value 10 as well (once you set index 1 to 10, setting index 2 to equal index 1 will set 2 to 10 as well, and so on).

Finally, it sets index 0 to tmp, which is 50

Thus, you end up with [ 50 ] [ 10 ] [ 10 ] [ 10 ] [ 10 ]

**Problem 2:**

import java.awt.Color;

import java.awt.event.\*;

import acm.graphics.\*;

import acm.program.\*;

import javax.swing.\*;

public class EtchASketch extends GraphicsProgram {

private static final double LINELENGTH = 10;

private static final double INCREMENT = 20;

private static final String NORTHTEXT = "North";

private static final String SOUTHTEXT = "South";

private static final String EASTTEXT = "East";

private static final String WESTTEXT = "West";

public void init() {

north = new JButton(NORTHTEXT);

south = new JButton(SOUTHTEXT);

east = new JButton(EASTTEXT);

west = new JButton(WESTTEXT);

add(north, SOUTH);

add(south, SOUTH);

add(east, SOUTH);

add(west, SOUTH);

drawX(getWidth()/2,getHeight()/2);

addActionListeners();

}

public void actionPerformed(ActionEvent e){

String cmd = e.getActionCommand();

moveX(cmd);

}

private void drawX(double x, double y){

currentX = x;

currentY = y;

cross = new GCompound();

GLine lineDn = new GLine(x- LINELENGTH /2,y- LINELENGTH /2,x+ LINELENGTH /2,y+ LINELENGTH /2);

GLine lineUp = new GLine(x- LINELENGTH /2,y+ LINELENGTH /2,x+ LINELENGTH /2,y- LINELENGTH /2);

cross.add(lineDn);

cross.add(lineUp);

add(cross);

}

private void moveX(String direction){

double dx = 0;

double dy = 0;

if(direction.equals(NORTHTEXT)){

dy = -INCREMENT;

} else if (direction.equals(SOUTHTEXT)){

dy = INCREMENT;

} else if (direction.equals(EASTTEXT)){

dx = INCREMENT;

} else if (direction.equals(WESTTEXT)){

dx = -INCREMENT;

}

remove(cross);

drawConnection(currentX+dx,currentY+dy);

drawX(currentX+dx,currentY+dy);

}

private void drawConnection(double newX, double newY){

GLine connection = new GLine(currentX,currentY,newX,newY);

connection.setColor(Color.RED);

add(connection);

}

/\* ivars \*/

private GCompound cross;

private double currentX;

private double currentY;

private JButton north;

private JButton south;

private JButton east;

private JButton west;

}

**Problem 3:**

import acm.program.\*;

public class WordLadder extends ConsoleProgram {

public void run() {

lexicon = new Lexicon("english.dat");

println("Program to check a word ladder.");

println("Enter a sequence of words ending with a blank line.");

while(true){

temp = readLine("");

if(temp.equals("")){

play = false;

break;

}

temp = temp.toUpperCase();

if(firstIsValid(temp)){

word=temp;

break;

} else {

println("That word is not legal. Try Again");

}

}

while(play){

String temp = readLine("");

if(temp.equals("")){

break;

}

temp = temp.toUpperCase();

if(isValid(temp)){

word = temp;

} else {

println("That word is not legal. Try Again.");

}

}

}

private boolean firstIsValid(String check){

if (!lexicon.isEnglishWord(check))return false;

return true;

}

private boolean isValid(String check){

if (!lexicon.isEnglishWord(check))return false;

if (check.length()!=word.length())return false;

if (!onlyOneChar(check))return false;

return true;

}

private boolean onlyOneChar(String check){

int count = 0;

for (int i = 0; i<check.length();i++){

if (check.charAt(i)!=word.charAt(i)) count++;

}

if(count==1)return true;

return false;

}

/\* ivars \*/

private Lexicon lexicon;

private String temp;

private String word;

private boolean play = true;

}

**Problem 4:**

private boolean checkUpperLeftCorner(int[][] matrix){

HashMap<Integer,Integer> counts = new HashMap<Integer,Integer>();

for (int i = 0; i<3;i++){

for (int j = 0; j<3; j++){

if(matrix[i][j]>9||matrix[i][j]<1)return false;

if(counts.get(matrix[i][j])==null){

counts.put(matrix[i][j], 1);

} else {

int count = counts.get(matrix[i][j]);

count++;

counts.put(matrix[i][j],count);

}

}

}

Iterator<Integer> it = counts.values().iterator();

while (it.hasNext()){

if (it.next()!=1)return false;

}

return true;

}

**Problem 5:**

import java.util.\*;

public class StringQueue implements MinimalStringQueue{

public StringQueue(){

queue = new ArrayList<String>();

}

public void add(String item){

queue.add(item);

}

public int size(){

return queue.size();

}

public String poll(){

if (queue.size()==0)return null;

String item = queue.get(0);

queue.remove(0);

return item;

}

/\* ivars \*/

ArrayList<String> queue;

}

**Problem 6:**

public boolean isGooglewhack(String w1, String w2){

pages = new HashMap<String,Integer>();

String[] pages1 = googleSearch(w1);

String[] pages2 = googleSearch(w2);

addPages(pages1);

addPages(pages2);

return checkPages();

}

private void addPages(String[] pgs){

for (int i = 0; i <pgs.length; i++){

int val;

if (!pages.containsKey(pgs[i])){

val = 1;

} else {

val = pages.get(pgs[i]);

val++;

}

pages.put(pgs[i],val);

}

}

private boolean checkPages(){

Iterator<Integer> it = pages.values().iterator();

int count = 0;

while(it.hasNext()){

int val = it.next();

if(val==2)count++;

}

if (count==1){

return true;

} else {

return false;

}

}

/\* ivars \*/

private HashMap<String,Integer> pages;

**Problem 7:**

public int commonKeyValuePairs(HashMap <String,String> map1, HashMap<String,String> map2) {

int count = 0;

Iterator<String> it1 = map1.keySet().iterator();

while(it1.hasNext()){

String first = it1.next();

String full = first+map1.get(first);

if(map2.containsKey(first)){

String first2 = first;

String full2 = first2+map2.get(first2);

if (full.equals(full2)) count++;

}

}

return count;