CS100M
Prelim 3

November 14, 2000
7:30 PM – 9:00 PM

____ ________________

(Print last name, first name, middle initial/name) (Student ID)

Statement of integrity: I did not, and will not, break the rules of academic integrity on this exam:

____

(Signature)

Circle Your Section:

	Tuesday			Wednesday	
	PH 403	PH 407	UH 111	HO 306	UH 111
1:25	1 Nagarajan		2 Fan	6 Rohde	
2:30		3 Nagarajan	5 Fan		7 Fernandes
3:35		4 Fernandes		8 Rohde	

Instructions:

- Read all instructions *carefully*, and read each problem *completely* before starting it!
- This test is closed book – no calculators, reference sheets, or any other material allowed.
- Conciseness, clarity, and style all count. Show all work to receive partial credit, especially box diagrams.
- Carefully comment each loop and major variable.
- If *you* use **`break`** or **`System.exit`** to exit any control structure (except **`switch`**), you will lose points!
- You may **not** use Java arrays or any MATLAB code.
- You may **not** alter, add, or remove any code that surrounds the blanks and boxes.
- Only **one** statement, expression, modifier, type, or comment per blank!
- Use the backs of pages if you need more space or scrap. You may request additional sheets from a proctor.
- If you supply multiple answers, we will grade only **one**.

Core Points:

```
    1. ________  (17 points) __________

    2. ________  (43 points) __________

    3. ________  (40 points) __________

Total: ________/(100 points) __________
```

Bonus Points:

```
       ________/(5 bonus points) ______
```

Problem 1 [17 points] *Code tracing, box scope diagrams*

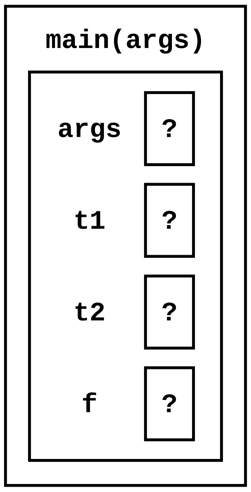
Complete the box scope diagrams up to the point indicated inside the following code that contains classes **`Problem1`**, **`Trewl`**, and **`Flurp`**. To help you out, do the following:

- Assume that all three classes belong to the same project.
- Use **only** the boxes we've drawn for you, including one box for the activation of the **`yalpu`** method.
- Do not draw boxes for constructor activations.
- Make sure that you fill in all initial values for instance and class variables.

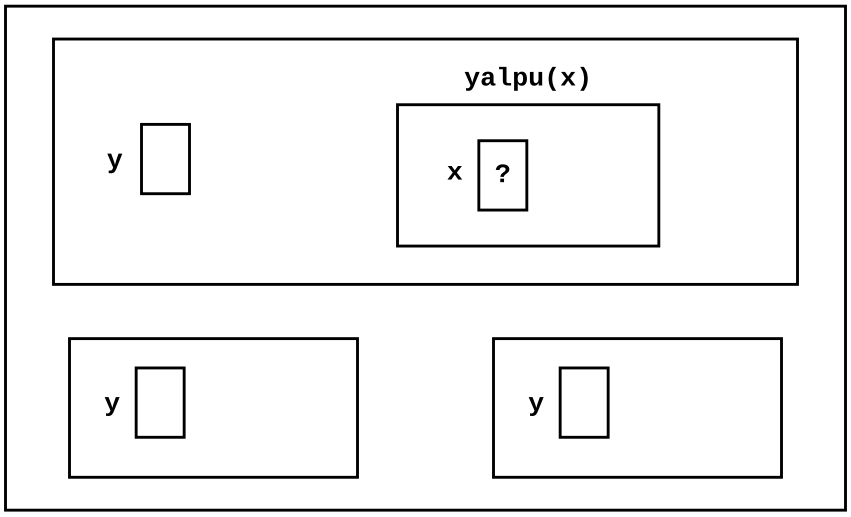
See the next page for spare boxes, in case you need to redraw your solution. **You must indicate which diagram we should grade!** Otherwise, we will choose the first marked diagram for grading.

```
public class Problem1 {
   public static void main(String[] args) {
      Trewl t1 = new Trewl();
      Trewl t2 = new Trewl();
      t1.yalpu(t2).w = 10;
      Flurp f = new Flurp();
      f.w = t2.y;
      f.z = t2;

      // Complete diagrams up to this point.

   }
}
```

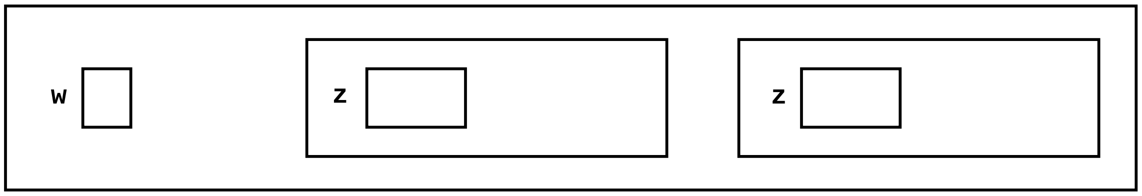
```
class Trewl {
   int y;
   Flurp yalpu(Trewl x) {
      x.y = 2;
      x = new Trewl();
      x.y = 3;
      return new Flurp();
   }
}
class Flurp {
   static int w;
   Trewl z;
}
```



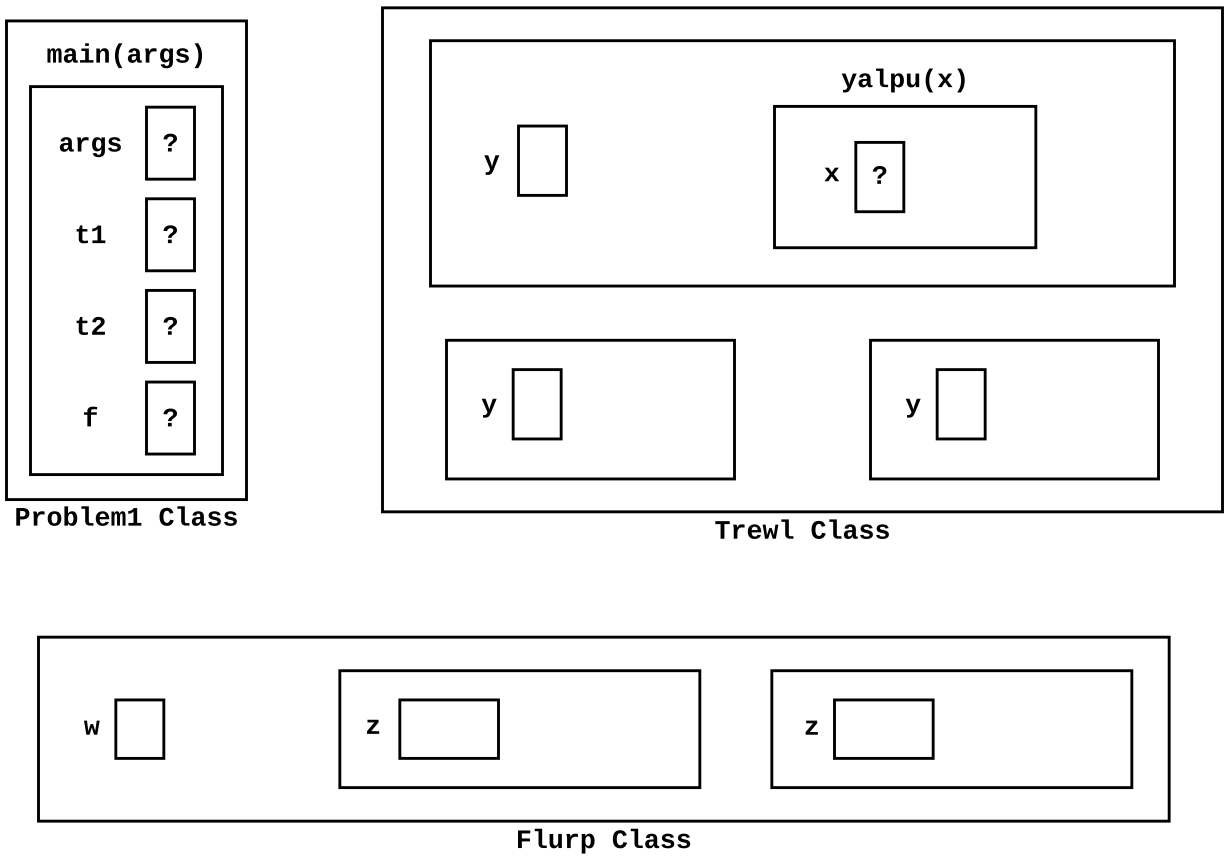
Problem1 Class

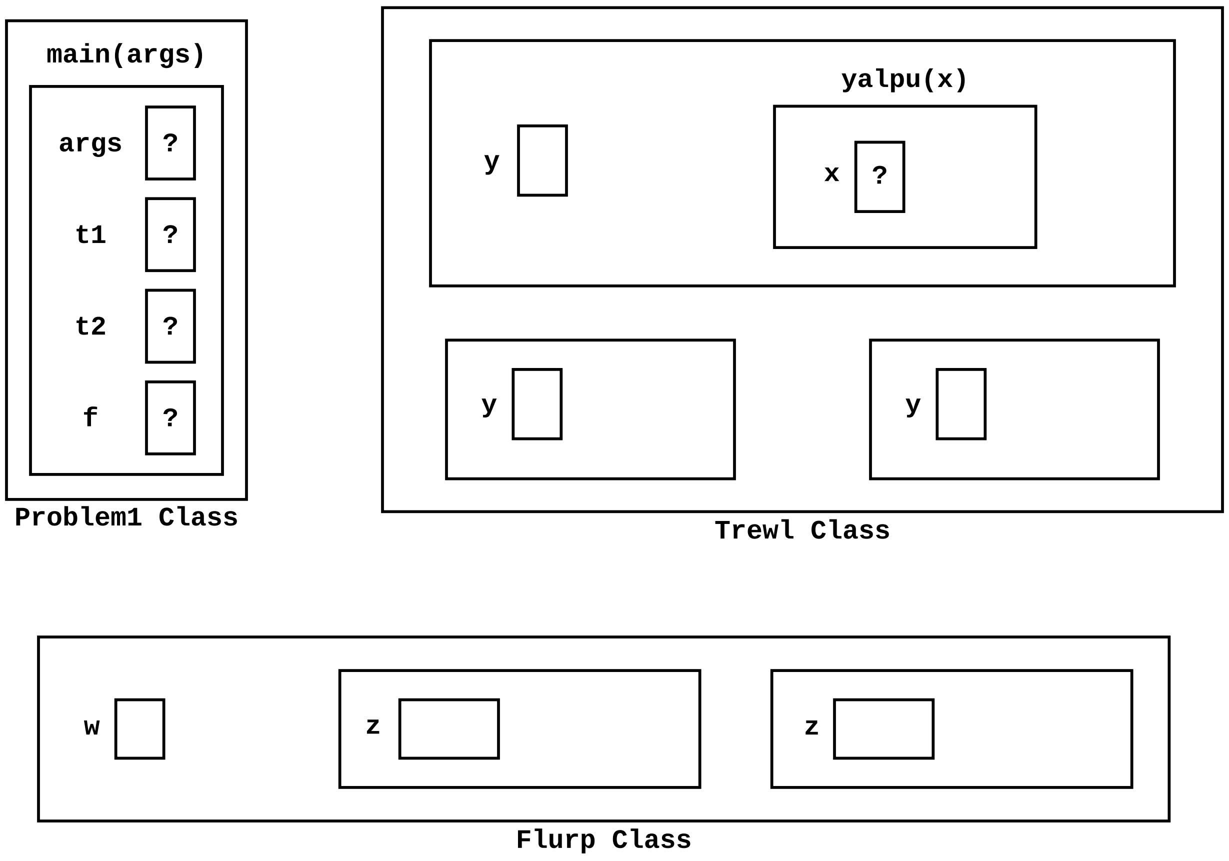


Trewl Class



Flurp Class


main(args)
args ?
t1 ?
t2 ?
f ?
Problem1 Class
yalpu(x)
y
x ?
y
y
Trewl Class
w
z
z
Flurp Class


main(args)
args ?
t1 ?
t2 ?
f ?
Problem1 Class
yalpu(x)
y
x ?
y
y
Trewl Class
w
z
z
Flurp Class

Problem 2 [43 points] *OOP, encapsulation,* **`this`**, *methods, conditions*

Complete the following code in class **`Length`** by filling in the blanks and boxes. **`Length`** represents a measurement of length in inches (in) or centimeters (cm). This class has the following members:

- instance variable **`numlen`** that stores the *numerical length*.
- instance variable **`units`** that stores the *unit label* as a **`String`**, which may be **`"in"`** or **`"cm"`**.
- constructor **`Length`** that sets **`numlen`** and **`units`**.
- instance method **`convert`** that takes as input a **`Length`** and a supplied unit label, compares the input **`Length`**'s **`units`** with the supplied unit label (using Java's **`equals`** method), and does the following, if necessary:
  - sets the input **`Length`**'s **`numlen`** to the numerical length in terms of the supplied unit label
  - sets the input **`Length`**'s **`units`** to the supplied unit label

  You do not need to account for illegal units. Hint: 1 in = 2.54 cm.
- instance method **`add`** that takes as input a **`Length`** and a supplied unit label and returns the sum of the input **`Length`** and current **`Length`** as a new **`Length`** in the units specified by the supplied unit label. Method **`add`** must call **`convert`** to perform any necessary unit conversions.
- instance method **`toString`** that returns a **`String`** containing the current **`numlen`** and **`units`**.

Note: In the box for **`convert`**, you must supply *brief* comments. We have supplied a Main Class that uses **`Length`** to add two measurements and convert the result to **`"cm"`**. The output for method **`main`** is `2.0 in + 3.0 cm = 8.08 cm`.

```
public class Problem2 {
   public static void main(String[] args) {
      // Create two Lengths:
         Length x1 = new Length(2,"in");  // 2 inches
         Length x2 = new Length(3,"cm");  // 3 centimeters
      // Add $x1$ and $x2$ together and report result in cm:
         System.out.println( x1 + " + " + x2 + " = " + x1.add(x2,"cm") );
   }
} // Class Problem2

class Length {

   ______________ double numlen; // numerical length

   ______________ String units;  // units of length

   // Construct a Length and set values of $numlen$ and $units$:

      ______________ Length(double numlen, String units) {

         ______________ = ______________ ;

         ______________ = ______________ ;

      }

   // Add current Length to another Length $x$. Return the sum as a new Length:

      ____________  Length add( ______________ x , ______________ choice ) {

         convert( ____________ , ______________________ ) ;

         convert( ____________ , ______________________ ) ;

         __________________________________________________________________ ;

      }
   // Convert input Length’s $numlen$ to the supplied unit label $choice$ and
   // update instance variables, if necessary.
      ____________  ____________ convert(Length x, String choice) {










      }

   // Return a String that contains the current object’s $numlen$ and $units$:

      ______________  _____________ toString() {

         return _______________________________________________ ;

      }

} // Class Length
```

Problem 3 [40 points] *OOP, encapsulation,* **`static`**, *constants, methods, flow control*

Background: A program can simulate the stacking of boxes on carts. Boxes of random height between 1 and 3 ft are stored on 3 carts that each have a height of 7 ft (feet). Boxes are stacked on top of each other on each cart, one cart at a time. On each cart boxes are stacked until their total height reaches as close as possible to, without exceeding, the top of the cart.

Problem: We have supplied the Main Class **`Problem3`** which drives the simulation, class **`Box`** which models the boxes, and some of class **`Cart`** which models the carts. You need to complete class **`Cart`** by filling in the blanks and boxes.

Approach: Complete the following methods that use the code we have provided for you:
- constructor **`Cart`** sets the current **`CartNumber`** and calls other instance methods to fill the current **`Cart`** and report the number of **`Box`**es stored on it.
- instance method **`fillOneCart`** stores **`Box`**es on the current **`Cart`** by instantiating new **`Box`**es until their height exceeds **`MAXHEIGHT`**. Creating a new **`Box`** simulates the stacking of the **`Box`** on top of the current **`Cart`**. Each time a **`Box`** is created, **`boxes`** increments by one to count the number of **`Box`**es on one **`Cart`**, so far. When no more **`Box`**es can fit on the current **`Cart`**, **`totalBoxes`** increments by **`boxes`** to count the total number of **`Box`**es for all **`Cart`**s, so far.
- class method **`fillAllCarts`** creates **`Cart`**s one at a time. When finished stacking the **`Box`**es on each **`Cart`**, **`fillAllCarts`** reports the total number of **`Box`**es stored on all **`Cart`**s.

Hint: Sample output has the following form (except the numbers of boxes might be different):

```
There are 2 boxes on Cart #1
There are 4 boxes on Cart #2
There are 3 boxes on Cart #3
There are a total of 9 boxes.
```

Notes: Remember that you may *not* use arrays! Avoid redundant/unnecessary code for full credit.

```
public class Problem3 {
   public static void main(String[] args) {
      Cart.fillAllCarts(); // Determine the total number of Boxes stored on all Carts
   }
} // Class Problem3

class Box {
   private double height;                          // height of Box
   public  double getHeight() { return height; }  // return height of current Box
   // Construct a Box with a random $height$ between 1 and 3 ft:
      public Box() { height = Math.random()*2+1; }
} // Class Box

class Cart {
   private int cartNumber;                 // Cart number
   private int boxes;                      // number of boxes stored on Cart
   private static int totalBoxes;          // total number of Boxes for all Carts
   private static int totalCarts;          // total number of Carts
   public  static final int MAXHEIGHT = 7; // max allowable height of Boxes on Carts
   public  static final int MAXCARTS  = 3; // maximum number of Carts

   // Construct a Cart:
   // assign Cart number, fill the current Cart with Boxes, and report contents:
      public Cart(int cartNumber) {

         _______________________________________ ; // assign CartNumber

         _______________________________________ ; // fill current Cart with Boxes

         _______________________________________ ; // report number of Boxes in Cart
      }
      // Put as many Boxes as possible in current Cart:
         private void fillOneCart() {




         }

   // Report the final number of Boxes stored on the current Cart:
      private void reportOneCart() {
         System.out.println("There are " + boxes + " boxes on Cart #" + cartNumber);
      }

   // Report the total number of Boxes on all Carts:
      private static void reportAllCarts() {
         System.out.println("There are a total of " + totalBoxes + " boxes.");
      }

   // Stack Boxes on each Cart, one Cart at a time:
      public static void fillAllCarts() {




      }

} // Class Cart
```

<u>Checklist</u>: Congratulations! You reached the last page of Prelim 3. Make sure your name, ID, and section are CLEARLY indicated. Also, re-read all problem descriptions/code comments/instructions. If you reached this part before exhausting the allotted time, check your test! Have you done the following?

- Completed all tasks
- Filled in ALL required blanks
- Given comments when necessary
- Declared all variables
- Maintained case-sensitivity
- Handled “special cases” correctly
- Indicated which solution to grade if you wrote multiple attempts

---

Bonus: [5 points] Remember that bonus points do not count towards your core-point total! You will lose additional points from your *entire* CS100M bonus score for “inappropriate” language.

[2 Bonus points] Why does DIS use only one public class per file in many of his examples?

[3 Bonus points] Why is Java’s method **`main`** modified as **`public static void`**?

`public`:

`static`:

`void`: