APCS Test 1 Review Packet:

TOPICS:
**Data types, arithmetic and various operators**
int, double, Boolean

Arithmetic operators: +, -, *, /, %
Assignment operators: =, +=, -=, *=, /=
Comparison operators: <, >, <=, >=, ==
Boolean operators: &&, ||, !

See Quiz 1

NOTES and SAMPLES:
- You can expect at least one problem where integer division and its effects will come into play. For example:
  ```java
  int a = 5, b=7;
  double c = (a/b) + 2;
  c in this case is equal to 2.0. Since (a/b) performs integer division it evaluates to 0. Then 2 is added and the result is assigned to a double, making it 2.0.
  ```
- You should know what the modulo operator (%) does even though we haven’t used it much yet. Modulo returns the remainder after integer division. EXAMPLE:
  ```java
  int a = 5, b =7;
  int c = a % b; //c = 5;
  c = b % a; //c = 2
  ```
- You should know that the comparison operators evaluate to Boolean values. e.g.
  ```java
  boolean c = a < b; is a valid assignment.
  ```
- You should have reasonable facility evaluating compound Boolean expressions. This means you should know the truth tables for && and ||. For example, I may ask whether the following expression evaluates to true or false depending on the values of a and b:
  ```java
  !(a < 5 || b > 9)
  ```
  Try different values for a and b to see what you get. EXAMPLE: if a = 6 and b = 6, the expression evaluates to TRUE.

**Stepwise-refinement**
See Quiz 2
There will likely be a problem that asks you to DESIGN but not write a class for robot that solves a certain problem. A design question is really an attempt to get to the heart of solving a problem without worrying about the minutia of code. As such, the names of the methods that you come up with are crucial to indicating that you know how to solve the problem. DO NOT be afraid of descriptive (or long) method names.
**Inheritance and Polymorphism**

See Quiz 3

You know the deal here. The question that tripped everybody up on Quiz 4 is about the toughest thing I could ask, so make sure you know it and understand it well.

NOTES:
- You should be very comfortable with extending classes and overriding methods and knowing what that means. This includes understanding an inheritance tree and knowing which methods you inherit from where.
- Understanding the difference between compile-time and run-time checking will aid your understanding.
- You WILL be asked to extend UrRobot, or Robot, and override one of its methods. Remember that if you want to override, say, the move() method, and you need to access the original definition, you need to call super.move() to get at it.

**If-else statements**

See class notes, and labs

We haven’t had a quiz on this yet, but after Carpenter Bot you should be well prepared. Be aware of the following:

NOTES and SAMPLES:
- If you have several consecutive if-statements without else clauses, each condition in EVERY if statement will be checked so there is a possibility that more than one condition could be true and there is a possibility that none could be true. Thus, there is no guarantee that any if-statement will be entered. For example, in the following lines of code, it’s possible that none of the if-blocks will be executed (if a = 11, for example) or more than one will be (if a = 4, for example).

```java
if(a < 5 ){
    System.out.println(“less than 5”);
}
if( a < 10){
    System.out.println(“less than 10”);
}
if(a > 100){
    System.out.println(“greater than 100”);
}
```
• Things change slightly if you change the code above to a series of else-ifs. Now, there’s still a change that no if-block will be entered, but if one is entered, ONLY ONE will execute. For example: if a=11, it is still the case that no condition is true so nothing will happen, but if a=4 then ONLY the first if-block will execute and all the others will be ignored:

```java
if(a < 5 ){
    System.out.println("less than 5");
}
else if( a < 10){
    System.out.println("less than 10");
}
else if(a > 100){
    System.out.println("greater than 100");
}
```

• The only way to guarantee that some block will be entered is to place a final else clause at the end of everything. In the code below one of the blocks of code will be executed no matter what the value of a is:

```java
if(a < 5 ){
    System.out.println("less than 5");
}
if( a < 10){
    System.out.println("less than 10");
}
if(a > 100){
    System.out.println("greater than 100");
}
else{
    System.out.println("a is between 11 and 100, inclusive");
}
```

**Recursive methods:**
When writing a recursive method, make sure to have some condition in it that does not invoke a recursive call. This is usually called the stopping case. Sometimes this condition is implied in the case where the method does nothing. For example, the stopping case is implied in the following method since the method won’t do anything if the robot is already facing north:

```java
public void turnNorth(){
    if(! facingNorth()){
        turnLeft();
    }
}
```

**Writing methods that return values**
We haven’t done a lot of this, but you should feel prepared to write methods that return Boolean values. For example, could you write the following method?

    public boolean beeperToRight(){ … }

It is usually understood that after a Boolean method has executed, the robot will be in the same state s/he started in before the method was called. Boolean methods give an answer to a question about the state of things in the world, they are not directives to perform any action…unless it’s clear from the name of the method itself. For example, you could write the following method to move the robot forward if the front is clear and return true, or do nothing and return false if it can’t move:

    public boolean moveIfClear(){
        if(frontIsClear()){
            move();
            return true;
        }
        else{
            return false;
        }
    }
Practice Problems

APCS 2008-2009

What follows are questions from past tests I’ve given – some tests are just copied verbatim. Don’t be confused by page headers, question numbers, etc. They’re all discombobulated.

A great way to review is to try to actually write some of these programs.

Note for 08-09: we have not explicitly discussed the three kinds of programming errors and you are not responsible for knowing the terminology. But just so you know, the kinds of errors are: A.) syntax – you forget a semi-colon, curly braces don’t match up, etc. B.) Execution errors – something that goes wrong when the program is running. And there are two different types of execution errors: 1. You do something that causes the program to crash – infinite recursion, sending a robot into a wall, etc. 2. The program works and finishes, but doesn’t do what it’s supposed to do – these are called logic errors.

Name:_____________________

NOTE: There are four numbered questions each worth 25% of your total score. When asked to write a class definition elegance does count for something. It is expected that you will use step-wise refinement and method factoring to arrive at your final solution. Use the backs of pages if you need to.

1.) (25 points)
a.) What is the difference between an algorithm and program?
b.) Give an example of each of the three kinds of programming errors. Please write the type of error and then an example.

c.) Given the following class header:

```java
public class Cappuccino extends Coffee
```
Why does one say that a Cappuccino is a Coffee?
2.) Write a complete class definition for a new Robot called Figure8Bot that is a sub-class of UrRobot. The Figure8Bot should have a constructor that simply passes its parameter values “up” to the constructor for UrRobot. Figure8Bot should override the move() method. A Figure8Bot moves in the shape of an 8 according to the diagram below:

(That is, the robot’s first move is forward, then right, forward, left, forward, etc..) You may include other helper methods if you like. Be careful about overriding the move() method. Write your class below.
3.) Meet Karel the gardener. Karel’s task is to plant a beeper on each location around each of the walls. The robot should start with 48 beepers in his bag. You may start him wherever you like. You will write code for both the Gardener robot class (part a) and a class with a main method to test it (part b). The “before” and “after” diagrams are shown here:

![Diagram showing the starting and ending situations.](image)

The starting situation

The ending situation –all beepers have been planted

a.) Write a class called GardenerBot which should be a sub-class of UrRobot. GardenerBot should have a constructor that simply passes its parameter values “up” to the constructor of UrRobot. GardenerBot can have whatever methods you like, but you should strive for an efficiently coded solution. You may want to start by thinking about what the main method will look like – what methods do you want to be able to call on your bot? Then use stepwise refinement to work your solution down to its base-level methods.
3.a. Con’d).
3.b.) Write a full class definition with a main method that uses GardenerBot. Be sure to make the World visible and to start your robot on a location, and facing a direction, that will work with your code. Then call whatever methods you need to make your GardenerBot work.
4.) What if the garden looked like the one pictured here instead.

If you had to write a new Robot class called DiamondGardener, how would you do it?

Try to think of a solution that doesn’t require “reinventing the wheel.” That is, think of a solution that would involve as little new code as possible. Please draw a UML diagram that illustrates your solution and write a brief description of how it will solve the problem.

(For extra credit you can write code for the whole class definition for DiamondGardener.)
2.) 20 points
Assume there exists a Robot called MathBot that finds numbers in the world (piles of beepers) and performs some mathematical operations on them. One of these methods is called absDiff which returns the absolute value of the difference between two numbers.

To be clear you are writing JUST the absDiff method which accepts two values, and returns a value – that’s it. The method doesn’t make the robot do anything. You need to write absDiff according to the following specifications:

- absDiff returns an int, and accepts two ints as parameters.
- it subtracts one number from the other and returns the absolute value (HINT: if you need to “flip” a number’s sign, you can simply multiply it by -1)

Write the full method definition, including the header, for absDiff as it might appear in the MathBot.
3.) 30 points
A Robot called PathFollower is supposed to be able to follow a path of beepers. You are going to write (override) the move method for PathFollower which will make PathFollower take one step along the path. When move is called, it finds the next place it’s supposed to move by checking it’s front, left, and right (but not back) positions for the next beeper in the path and then moves there. If there is no beeper to the front, left or right, it should simply move forward.

A PathFollower might be used in a main program in the following way.

```
public class PathTest{
    public static void main (String [] args){
        // <...World loading and setup not shown...>
        PathFollower P = new PathFollower(2,2,Directions.East,0);
        while(P.nextToABeeper()){
            P.move();
        }
        P.turnOff();
    }
}
```

Write the move method for PathFollower. If you like, you may write supporting methods if you think it will make your move code clearer. It is started for you below (on the next page):

```
public class PathFollower extends Robot{
    <...typical constructor not shown ...>
    public void turnaround(){
        turnaround(); turnLeft();
    }
    public void turnRight(){
        turnaround(); turnLeft();
    }
    public void move(){
```