# CS46A Review

**Directions:** All problems refer to the Java programming language. Provide complete but concise answers. If you do not have enough space for your answer, use the back of the previous page.

1)

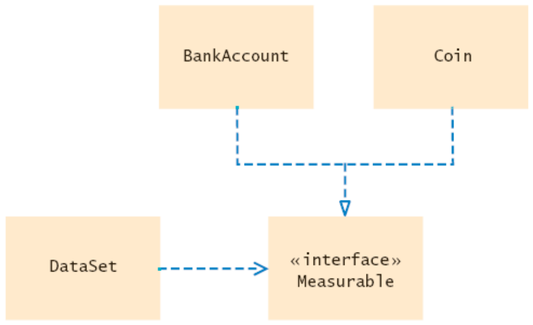
a) Write a precondition for the following. Do not implement the method.

```
public static double sqrt(double x)
```

b) What side effect, if any, does the following method have?

```
public void transfer(double amount, BankAccount other)
{
        balance = balance – amount;
        other.balance = other.balance + amount;
}
```

2) The UML diagram below indicates that `BankAccount` "is-a" `Measurable` and `Coin` "is-a" `Measurable`, while `DataSet` "uses" `Measurable`.



a) Of the three classes in this diagram, which implement the `Measurable` interface and which do not?

b) What are the two most significant benefits of defining an interface?

3) True or false

   a) _____ A static field belongs to the class, not to a particular instance of the class.

   b) _____ A static method has no implicit parameter.

   c) _____ The `superduper` keyword is used to call a method of the superclass.

   d) _____ To call the superclass constructor, you use the `super` keyword in the first statement of the subclass constructor.

   e) _____ A subclass must inherit any `final` methods from the corresponding superclass.

4) Consider the following program that accepts a set of integers from a user and determines and prints to console only the even numbers. Complete missing Java statements in the main method so that if a user types:

```
1
4
5
2
q
```

The program prints:

```
4
2
```

```
public static void main(String[] args)
{
   ArrayList<Integer> numbers = new ArrayList<Integer>();
   Scanner read = new Scanner(System.in);
   boolean done = false;
   while (!done)
   {
      //your code goes here







  }
```

5) Consider the `AccountTester` class below.

```
class AccountTester
{
    public static void main(String [] args)
    {
        SavingsAccount momsSavings = new SavingsAccount(0.5);
        CheckingAccount harrysChecking = new CheckingAccount(0);
        . . .

        endOfMonth(momsSavings);
        endOfMonth(harrysChecking);
        printBalance(momsSavings);
        printBalance(harrysChecking);
    }
    public static void endOfMonth(SavingsAccount savings)
    {
        savings.addInterest();
    }
    public static void endOfMonth(CheckingAccount checking)
    {
        checking.deductFees();
    }
    public static void printBalance(BankAccount account)
    {
        System.out.println("The balance is $"
            + account.getBalance());
    }
}
```

a) Define early binding and late binding.

b) Are the calls to `endOfMonth` resolved by early binding or late binding? Explain.

c) Inside `printBalance`, is the call to `getBalance` resolved by early or late binding? Explain.

6) Consider the `BankAccount` and `BankAccountTester` classes below.

```
public class BankAccount
{
    . . .
    public static void swap(BankAccount a, BankAccount b)
    {
        BankAccount temp = a;
        a = b;
        b = temp;
    }
    . . .
    private double balance;
}
class BankAccountTester
{
    public static void main(String [] args)
    {
        BankAccount x = new BankAccount(5000);
        BankAccount y = new BankAccount(10);
        System.out.println("Before swap: x balance = $"
            + x.getBalance());
        System.out.println("Before swap: y balance = $"
            + y.getBalance());
        BankAccount.swap(x, y);
        System.out.println("After swap: x balance = $"
            + x.getBalance());
        System.out.println("After swap: y balance = $"
            + y.getBalance());
    }
}
```

a) The following output was obtained from `BankAccountTester`. Explain why the balances of `BankAccount x` and `BankAccount y` were not swapped.

```
Before swap: x balance = $5000.0
Before swap: y balance = $10.0
After swap: x balance = $5000.0
After swap: y balance = $10.0
```

b) Modify the `swap` method of `BankAccount` so that the bank account balances are swapped. Your program should produce the following results:

```
Before swap: x balance = $5000.0
Before swap: y balance = $10.0
After swap: x balance = $10.0
After swap: y balance = $5000.0
```

7) Use the `Measurable` interface and the `DataSet` class below to do the following: Define a class `Car` that implements the `Measurable` interface, where a car has a make and model (for example, "Honda Civic") and a sale price. Use the `DataSet` class to process a collection of cars. Provide a test program that creates a data set with five cars. Your test program should display the make, model and selling price of the most expensive car, as well as the average price of the cars in your data set.

```
public interface Measurable
{
   double getMeasure();
}
public class DataSet
{
   public DataSet()
   {
      sum = 0;
      count = 0;
      maximum = null;
   }
   public void add(Measurable x)
   {
      sum = sum + x.getMeasure();
      if (count == 0 || maximum.getMeasure() < x.getMeasure())
         maximum = x;
      count++;
   }
   public double getAverage()
   {
      if (count == 0) return 0;
      else return sum / count;
   }
   public Measurable getMaximum()
   {
      return maximum;
   }
   private double sum;
   private Measurable maximum;
   private int count;
}
```

8) In each of the following program segments there is an error. There is only one error in each code segment. Your goal is to (a) identify an error, (b) classify the error as either compile-time error or run-time error and (c) explain how to fix the error.

```
String input = JOptionPane.showInputDialog("Enter an integer:");
int result = input * 2;
System.out.println(result);
```

a) Error:
b) Type of error:
c) Fix:

```
Random r = new Random();
String randmLetters = "";
String[] words = {"a", "b", "c"};
for(int i = 0; i < words.length; i++)
{
     randmLetters = randmLetters.concat(words[r.nextInt(3)+1]);
}
System.out.println(randmLetters);
```

a) Error:
b) Type of error:
c) Fix:

```
class RoachPopulation
{
     /**
          constructor of RoachPopulation class
          @param initialPopulation starting number of roaches
     */
     public RoachPopulation(double initialPopulation)
     {
          double population = initialPopulation;
     }
     /**
          breed method doubles the population size
     */
     public void breed()
     {
          population = population * 2;
     }
     /**
          spray method reduces the population size by 10%
     */
     public void spray()
     {
          population = population - population * 0.1;
     }
     /**
          getRoaches method retrieves current population size
          @return population size
     */
     public double getRoaches()
     {
          return population;
     }
     private double population;
}
```

a) Error:
b) Type of error:
c) Fix:

9) Consider the two classes below, `Insect` and `Bug`.
   a) Give the name(s) of the overloaded method(s):

   b) Give the name(s) of the overridden method(s):

```
public class Insect
{
     public Insect(double initPosition)
     {
          position = initPosition;
     }
     public void fly(double distance)
     {
          position += distance;
     }
     public String describe()
     {
          return "I am an insect";
     }
     public double getPosition()
     {
          return position;
     }
     private double position;
}

public class Bug extends Insect
{
     public Bug(String aType, double initPosition)
     {
          super(initPosition);
          type = aType;
     }
     public void move()
     {
          super.fly(1);
     }
     public void move(double steps)
     {
          super.fly(steps);
     }
     public String describe()
     {
          return super.describe() + " and I am a bug";
     }
     public String type;
}
```

10) Consider the following problem description: A shopping cart can have many products. Each product has a name and a price. Products can be added to the cart and a total number of products in a cart can be determined.

   a) Consider the implementation of the `Product` class below. Write javadoc comments for the constructor and `getName` method in the `Product` class.

```
public class Product
{






     public Product(String aName, double aPrice)
     {
          name = aName;
          price = aPrice;
     }

     public String getName()
     {
          return name;
     }






     public double getPrice()
     {
          return price;
     }

     private String name;
     private double price;
}
```

   b) Implement the shopping cart class based on the problem description