Solutions to OOP Exam, Spring 2005.

Question 1.a
When the object is created, i gets its default value. i is an integer, and the default value is 0. Next the initializers are executed once. In this case, i is assigned the value returned by foo, which is 8. Lastly, the body of the constructor is executed, and i gets the value 25 (15 passed as parameter + 10). The initialization order is described by Java Precisely at the end of page 28.

Question 1.b
The object created has two i-fields, one from class A and one from B. both will first get the default value 0. Then the initializers and contructor from A will be executed. The i field declared in A will be initialized to the value returned by foo(). foo has been redeclared in B, where it returns the *current* value of the i declared in B. This is 0. Thus the i declared in A is initialized to 0. Then the constructor in A is executed, with parameter 15+4. Thus, i declared in A is assigned 29.
Then the initializers and constructor in B are executed. The i declared in B gets the value 55. Then the body of the B constructor is executed. It sets the i declared in B to the value of the i declared in A + 3, that is 32.
The rules are the same as used in 1.a.

Question 1.c
It will print different numbers, namely 29 and 32, because fields are not virtual, so a.i refers to the field declared in the class which was used to declare a, in this case A. b is declared of type B, so a.i and b.i refers to the two different i fields.

Question 2.a
The invariant is already given, it is the two bullet points above question 2.a.

Question 2.b

```
MergeIterators(Iterator a, Iterator b){
    this.a = a;
    this.b = b;
}
public boolean hasNext(){ return a.hasNext() || b.hasNext(); }

public Object next(){
    if (a.hasNext() )
        if (b.hasNext())
            if ( Math.random() < 0.5 )
                return a.next();
            else
```

```
                return b.next();
            else
                return a.next();
        else
            return b.next();
    }
```

Question 3.a & b

```
public class Kindergarden implements Iterable<Child>{

    Set<Child> children = new HashSet<Child>();

    public Iterator<Child> iterator(){
        return children.iterator();
    }
    public void enroll(Child c){
        children.add( c );
    }

    public void withdraw(Child c){
        children.remove(c);
    }


}
```

Question 3.c

```
public class Institution<T extends Person> implements Iterable<T>{

    Set<T> associates = new HashSet<T>();

    public Iterator<T> iterator(){
        return associates.iterator();
    }
    public void enroll(T c){
        associates.add( c );
    }

    public void withdraw(T c){
        associates.remove(c);
    }
}
```

Question 3.d

```
public <P extends T> void  merge(Institution<P> other){
    for(P person: other) {other.withdraw( person); this.enroll( person );}
}
```

<…1…> declares the type parameter P to be used in the merge. P must be a subtype of the type of person T for this institution.

Question 4.a

```
public double readTemperature() throws SensorError{
   double t = prim_read();
   if (t == -300)
      throw new ReadError();
   else
      return t;
}
```

According to the exercise text, one should expect the reader to fail occasionally. This failure is outside the programmers' control. Thus, the programmer should be prepared to handle the situation that something goes wrong. This is the classical argument for using checked exceptions.

Question 4.b

```
public class TemperatureSensor{

   public double readTemperature() throws SensorError{
      double t = prim_read();
      if (t == -300)
         throw new SensorError();
      else
         return t;
   }

   class SensorError extends Exception {
   }

}
```

Question 4.c

```
public static void monitorTemperature(double low, double high){
   TemperatureSensor ts = new TemperatureSensor();
   int errorReadings = 0;
   while(true)
      try{
         double t = ts.readTemperature();
         errorReadings = 0;
         if (t<low)
            System.out.println("Low temperature: " + t);
         if (t>high)
            System.out.println("High temperature: " + t);
      }catch(SensorError re){
         errorReadings++;
         if (errorReadings>5){
            System.out.println("Cannot read the temperature");
            errorReadings = 0;
         }
      }
      try{ Thread.sleep(1000);}catch(InterruptedException ie){};
   }
}
```

Question 5.a

```
public class MyMonitor implements GeneralTemperatureMonitor.TemperatureObserver{
   public MyMonitor (GeneralTemperatureMonitor gtm){
      gtm.observers.add(this);
      readFailures = 0;
   }
   int readFailures;
   public void observed(double temp){
      readFailures=0;
      if (temp>25)
         System.out.println("Got large: " + temp);
      if (temp<15)
         System.out.println("Got small: " + temp);
   };

   public void noRead(){
      readFailures++;
      if ( readFailures >= 5){
         System.out.println("Failed reading 5 times");
         readFailures = 0;
      }
   };
}
```

Question 5.b

The TemperatureObserver interface is changed to:

```
public interface TemperatureObserver {
   void observed(double temperature);
   void noRead();
   boolean checkReading(double temp);
}
```

The body of the startMonitoring method is changed. The for loop

```
for(TemperatureObserver to:observers)
    to.observed( temp );
```

is changed to

```
for(TemperatureObserver to:observers)
    if (to.checkReading(temp) ) to.observed( temp );
```

so that we only report an observed temperature on given conditions.

Question 5.c

```
public classMyObserver2 implements GTM.TemperatureObserver{
   public classMyObserver2 (GTM gtm){
      gtm.observers.add(this);
      readFailures = 0;
   }
   public boolean checkReading(double temp){
      readFailures = 0;
      return (temp<15) || (temp>25);
   }
   int readFailures;
   public void observed(double temp){
      System.out.println("Got abnormal: " + temp);
   };

   public void noRead(){
      readFailures++;
      if ( readFailures >= 5){
         System.out.println("Failed reading 5 times");
         readFailures = 0;
      }
   };
}
```