Part I. Fill in the Blank. (10 points)

1. There were two conditions listed for a user-defined data type to be considered an Abstract Data Type, one of them is single syntactic unit OR information hiding (pg 430).

2. In Java, you can prevent a method in a class from being overridden by declaring that method to be final.

3. A data member that is shared among every class instance is called a(n) class variable . It is created in C++ and Java by declaring that data member to be static .

4. An ADT definition that takes a parameter indicating the type of data the ADT operates on is called a(n) parameterized type . In C++, you can create this with a(n) template . Can you even do this in Java (yes/no)? Yes (sort of).

5. Syntax is specified concretely using Backus-Naur Form (BNF) or E-BNF. There are three methods that are used to specify dynamic semantics; two of them are operational OR denotational OR axiomatic .

6. If a method in a child class has the same name, same number and same type of parameters as a method in its parent class then that method is said to be overridden .
Part II. Short Answer (24 points)

7. (4 points) The concept of naming encapsulations is implemented in C++ with the “namespace” construct and in Java with the “package” construct. Explain what a naming encapsulation provides to programmers. It allows us to group together related items (classes usually). It also allows us to re-use names; for example, a Node class can appear in different namespaces. The namespace can be used to disambiguate which Node is being referred to. (In Java, the package also defines certain access rights to class fields and methods.)

8. (4 points) One of the design issues of object oriented languages is the allocation and deallocation of objects. Choose a weapon, either Java or C++, and discuss how that language addresses this issue. C++: Allocation is done explicitly, constructors can be overloaded, objects can be created dynamically on the stack or in the heap. Destructors can be explicitly called. Java: Allocation is done explicitly, constructors can be overloaded, objects created dynamically only in the heap. Deallocation is performed implicitly by the garbage collector.

9. (4 points) In Java, each class is allowed to define a finalize() method. Discuss the drawbacks to using this approach to “clean up” any object allocated system resources (e.g., files). We can NOT control when this method is actually called since it is called during garbage collection. Indeed, this method may NOT be called at all!
10. (4 points) What is the difference between a synthesized and an inherited attribute in attribute grammars? Synthesized attributes get their values from “lower-to-higher” in the parse tree. That is, leaves intrinsically know their synthesized attribute values and the root will compute its synthesized attribute values LAST.

Inherited attributes get values top-down.

11. (4 points) One way to know if a sentence is in the language generated by a grammar is find a derivation for the sentence. Show a derivation to prove that the string \texttt{bbaab} is in the language defined by the grammar:

\[
\begin{align*}
\text{<S>} & \rightarrow \text{<A>} \ a \ \text{<B>} \ b \\
\text{<A>} & \rightarrow \text{<A>} \ b \ | \ b \\
\text{<B>} & \rightarrow \ a \ \text{<B>} \ | \ a
\end{align*}
\]

One derivation is (there are several possible ones):

\[
\begin{align*}
\text{<S>} & \rightarrow \text{<A>} \ a \ \text{<B>} \ b \\
& \rightarrow \text{<A>} \ b \ a \ \text{<B>} \ b \\
& \rightarrow \ b \ b \ a \ \text{<B>} \ b \\
& \rightarrow \ b \ b \ a \ a \ b
\end{align*}
\]

12. (4 points) \textbf{Choose your weapon: C++ or Java} (Extra credit for both!)

(a) (C++) Why should we use “initialization lists” instead of assignments in constructor bodies?

(b) (Java) Explain how an interface differs from a class.

\textbf{C++:} It often saves time by calling a parameterized constructor instead of a the default constructor, parameterized constructor, copy constructor sequence.

\textbf{Java:} Interfaces can NOT have instance variables (only constants) and do NOT define method bodies. They are used as behavior specifications.
Part III. Problems (66 points)

13. (8 points) This question tests your understanding of the inheritance relationship between classes as they are constructed. What is the output of the program below? (The C++/Java equivalent program is given, you only need to use one since the answer is the same!)

```java
public class Constructors {
    public static void main(String[] args) {
        LongHairedCat minerva;
        minerva = new LongHairedCat();
    }
}
```

```cpp
class Animal {
    public:
    Animal() {
        cout << "Animal" << endl;
    }
}
class Cat : Animal {
    public:
    Cat() {
        cout << "Cat" << endl;
    }
}
class LongHairedCat : Cat {
    public:
    LongHairedCat() {
        cout << "LongHairedCat" << endl;
    }
}
main() {
    LongHairedCat *minerva;
    minerva = new LongHairedCat();
}
Animal
Cat
LongHairedCat
```
14. (8 points) Write the recursive descent parsing functions/methods that correspond to the grammar below. (Assume you have a lexical analyzer named lex(), the integer variable nextToken, and the error() handler. You can also assume a “predefined” integer values with the names: a_Tok, b_Tok, c_Tok. So, you only have to write the A, B, and S methods.)

\[
\begin{align*}
&S \rightarrow a\ a\ <A> \mid b \mid c\ a\ <B> \\
&A \rightarrow a\ <B> \mid b \mid c\ <B>\ <B> \\
&B \rightarrow b\ <A> \mid a\ <B>\ b
\end{align*}
\]

S() {
    // nextToken always set up for functions (by main in this case)
    if (nextToken == a_Tok) {
        lex();
        if (nextToken == a_Tok) { lex(); A(); }  
        else error();
    }
    else if (nextToken == b_Tok) { lex(); }
    else if (nextToken == c_Tok) {
        lex();
        if (nextToken == a_Tok) { lex(); B(); }  
        else error();
    }
    else error();
}

A() {
    if (nextToken == a_Tok) { lex(); B(); }
    else if (nextToken == b_Tok) { lex(); }
    else if (nextToken == c_Tok) { lex(); B(); B(); }
    else error();
}

B() {
    if (nextToken == a_Tok) {
        lex();
        B();
        if (nextToken == b_Tok) lex();
        else error();
    }
    else if (nextToken == b_Tok) {lex(); A();}
    else error();
}
15. (9 points) This problem tests your understanding of polymorphism in C++. What is the output of the following program?

```cpp
using namespace std;
#include <iostream>
class X {
  public:
    virtual void foo() { cout << "foo in X called\n"; }
    void goo() { cout << "goo in X called\n"; }
    void hoo() { cout << "hoo in X called\n"; }
};
class Y : public X {
  public:
    virtual void foo() { cout << "foo in Y called\n"; }
    virtual void goo() { cout << "goo in Y called\n"; }
    void hoo() { cout << "hoo in Y called\n"; }
};
class Z : public Y {
  public:
    virtual void foo() { cout << "foo in Z called\n"; }
    virtual void goo() { cout << "goo in Z called\n"; }
    void hoo() { cout << "hoo in Z called\n"; }
};

int main() {
  X * xptr; Y *yptr; X xobj; Y yobj; Z zobj;
  xptr = &xobj;  // if not virtual in X then defaults to X
  xptr->foo();  foo in X called
  xptr->goo();  goo in X called
  xptr->hoo();  hoo in X called

  xptr = &yobj;
  xptr->foo();  foo in Y called
  xptr->goo();  goo in X called
  xptr->hoo();  hoo in X called

  xptr = &zobj;
  xptr->foo();  foo in Z called
  xptr->goo();  goo in X called
  xptr->hoo();  hoo in X called

  yptr = &yobj;  // if not virtual in Y then defaults to X
  yptr->foo();  foo in Y called
  yptr->goo();  goo in Y called
  yptr->hoo();  hoo in Y called

  yptr = &zobj;
  yptr->foo();  foo in Z called
  yptr->goo();  goo in Z called
  yptr->hoo();  hoo in Y called
}
```
16. (8 points) This problem tests your understanding of access permissions in C++ (in particular, with respect to inheritance). For each “numbered” statement below, indicate whether the statement will or will not cause a compilation error (just write “BAD” or “OK”).

```cpp
using namespace std;
#include <iostream>
class A {
    private:
        int a;
    public:
        int b;
    protected:
        int c;
};
class B : protected A {
    private:
        int d;
    public:
        int e;
    protected:
        int f;
    public:
    void foo() {
        a = 3; /* 1 */ BAD
        b = 3; /* 2 */ OK
        c = 3; /* 3 */ OK
        d = 3; /* 4 */ OK
        e = 3; /* 5 */ OK
        f = 3; /* 6 */ OK
    }
};
class C : public B {
    private:
        int g;
    public:
        int h;
    protected:
        int i;
    public:
    void foo() {
        a = 3; /* 7 */ BAD
        b = 3; /* 8 */ OK
        c = 3; /* 9 */ OK
        d = 3; /* 10 */ BAD
        e = 3; /* 11 */ OK
        f = 3; /* 12 */ OK
        g = 3; /* 13 */ OK
        h = 3; /* 14 */ OK
        i = 3; /* 15 */ OK
    }
};
```
17. (8 points) Recall that the area needed to execute a Java or C++ program can be considered four distinct areas: heap, stack, read/write segment, and read-only segment. Consider the following C++ code.

```cpp
class Environment {
    static int globalBugs;
    int localBugs;
    LinkedList lst;
    Environment() { localBugs = 0; }
    public void AddBug() {
        Bug * bug = new Bug("hairy");
        lst.Add(bug);
        localBugs++;
        globalBugs++;
    }
}
```

(a) Which of the data items (localBugs, globalBugs, lst, bug, the object pointed to by bug, the string "hairy") would fall into the heap area of memory?

(b) Which of the data items (localBugs, globalBugs, lst, bug, the object pointed to by bug, the string "hairy") would fall into the stack area of memory?

(c) Which of the data items (localBugs, globalBugs, lst, bug, the object pointed to by bug, the string "hairy") would fall into the read-only segment of memory?

(d) Which of the data items (localBugs, globalBugs, lst, bug, the object pointed to by bug, the string "hairy") would fall into the read/write segment of memory?

(e) Which of the data items (localBugs, globalBugs, lst, bug, the object pointed to by bug, the string "hairy") have locations that cannot be determined by simply looking at this small segment? Explain.
18. (9 points) This problem tests your understanding of Exception Handling in Java.

    foo() {
        try {
            goo();
            statement2;
        }
        catch (Exception_1 ex1) { statement3; }
        catch (Exception e) { statement4; }
        catch (Exception_2 ex2) { statement5; }
        finally { statement6; }
    }
    statement7;
}
    goo() {
        statement8;
        statement9;
    }

(a) If an exception occurs at statement8 and it is of type Exception_1, will the exception be caught? YES

(b) If an exception occurs at statement8 and it is of type Exception_1, will statement9 be executed? NO

(c) If an exception occurs at statement2 (and no exceptions occur prior to this one) and it is of type Exception_2, will statement3 be executed? NO

(d) If an exception occurs at statement2 (and no exceptions occur prior to this one) and it is of type Exception_2, will statement6 be executed? YES

(e) If an exception occurs at statement2 (and no exceptions occur prior to this one) and it is of type Exception_2, will statement7 be executed? YES

(f) If an exception occurs at statement2 (and no exceptions occur prior to this one) and it is not of type Exception_1 or Exception_2, will statement6 be executed? YES

(g) If an exception occurs at statement2 (and no exceptions occur prior to this one) and it is not of type Exception_1 or Exception_2, will statement7 be executed? YES

(h) An exception of type Exception_3 is raised in statement9. List all the statements executed (from the beginning of foo in the order that they are executed. 8, 9, 4, 6, 7
This problem tests your understanding of parameter passing in Java.

```java
class Integer {
    private int num;
    public Integer(int n) { num = n; }
    public Integer() {num = 0; }
    public void setInteger(int n) { num = n; }
    public int getInteger() {return num; }
}

class Example {
    public static void main(String args[]) {
        Integer i = new Integer(3);
        System.out.println("Value of i is " + i.getInteger());
        increment(i);
        System.out.println("Value of i is " + i.getInteger());
        increment(i);
        System.out.println("Value of i is " + i.getInteger());
    }
    public static void increment(Integer i) {
        Integer j = new Integer(i.getInteger() + 1);
        i = j;
    }
}
```

The output of this program is

```
Value of i is 3
Value of i is 3
Value of i is 3
```

This demonstrates that parameters in Java are passed by **VALUE**.

Modify the increment method so that the program would produce the expected output.

**Body of method should be:**

```
i.setInteger(i.getInteger()+1);
```
20. (8 points) This problem tests your understanding of static and final methods and data members in Java. Indicate which of the statements numbered below would cause a compilation error (just write “BAD” or “OK”).

class Problem {
    public static final int len = 20;
    public static int count = 10;

    public void foofun() {
        goofun(); /* 1 */ OK
        count++; /* 2 */ OK
        len = 200; /* 3 */ BAD
        Problem.goofun(); /* 4 */ OK
    }

    public static void goofun() {
        count = 100; /* 5 */ OK
        len = 200; /* 6 */ BAD
        foofun(); /* 7 */ BAD
        Problem.foofun(); /* 8 */ BAD
    }

    public static void main(String args[]) {
        Problem p = new Problem();
        p.foofun(); /* 9 */ OK
        foofun(); /* 10 */ BAD
        p.goofun(); /* 11 */ OK
        goofun(); /* 12 */ OK
    }
}