***Homework Problems 16***

1. What is the difference between an abstract class and an interface?
2. Why can an Object reference point to any object?
3. Write the statement that calls the f method in the object to which r points. Assume f takes no arguments, r is type A, and the object to which r points is type B. Assume the B class has an f method but the class A does not. B is a subclass of A.
4. Can an abstract class be extended?
5. Define a class that implements the following interface:

interface Wisp  
 {

public void f();

}

1. Comment out the xDisplay methods in the P and Q classes in C16h6.java (copy of Generic2.java in Fig. 16.4). Compile and run. What happens?
2. Comment out the xDisplay methods in the P and Q classes in C16h7.java (copy of Generic3.java in Fig. 16.5). Compile and run. What happens?
3. Comment out the xDisplay methods in the P and Q classes in C16h8.java (copy of Generic4.java in Fig. 16.6). Compile and run. What happens?
4. Write a program that consists of an interface Kons, and two classes C16h9 and Cusp, both of which implement Kons. The interface should contain the constants KON1 and KON2 equal to 1.3 and 5.5 respectively. main should display the values of KON1 and KON2. It should also create a Cusp object and call its display method. The Cusp class should contain a display method that displays the KON1 and KON2 constants. Use the file named C16h9.java.
5. Is the following code legal. If so, what happens when it is executed?

Object obj = "hello";  
 System.out.println(obj);

1. Does the following program compile and run? If not, fix it so that it does.

class C16h11

{

public static void main(String[] args)

{

Vincent v = new Vincent();

v.f();

v.g();

v.display();

}

}

//================================================

abstract class Dali

{

private int x = 1;

private static int y = 3;

public abstract void f();

public void g()

{

System.out.println("in g");

}

}

//================================================

class Vincent extents Dali

{

public void f()

{

Sytem.out.println("in f");

}

public void display()

{

System.out.println(x + " " + y);

}

}

1. Create a class Objects that contains the following fields:

private Object[] oa = new Object[2];

private int nextIndex = 0;

Objects should also contain the following methods:

public void add(Object obj)

Assigns obj to the next available slot in the oa array, and increments nextIndex. If there are no more available slots, add creates a new array with 10 more slots than the current oa array. It then copies the contents of the current oa to the new array. It finally assigns to oa the reference to the new array.

public Object get(int i)

Returns oa[i].

public void display()

Displays all the objects in oa by calling println for each object stored in the oa array.

public int size()

Returns nextIndex, whose value equals the number of occupied slots in oa.

Write a complete program in which main creates an Objects object to which it adds ten strings and ten Integer objects. main then calls the display method. Use the file C16h12.Java.

1. Create an interface named Shape that has two methods: public double getArea() and public void display(). Also create three classes that implement Shape: Circle, Square, and Rectangle. Each class should have a getArea method that returns the area of the object, and a display method that displays the dimensions of the object. In main, create an Object array with 12 slots to which you assign Circle, Square, and Rectangle objects whose dimensions are initialized with random numbers. Then execute a for loop that processes each reference in the Object array, displaying the dimension and area of each object. Use the file named C16h13.java.
2. Same as homework problem 13 except use a Shape array in place of the Object array. Use the file named C16h14.java.
3. Same as homework problem 13 except use an abstract class in place of an interface. Use the file named C16h15.java
4. Same as homework problem 13 except make Square a subclass of Rectangle. Use the file named C16h16.java
5. It makes sense for the String class have a toString method. Why?
6. Implement the toString method as it would appear in the String class.
7. Can a generic class have more than one type parameter. For example, is the following class legal? Use it in a program to see if it works.

class TwoParms<T, U>

{

T x;

U y;

//----------------------------------

public TwoParms(T xx, U yy)

{

x = xx;

y = yy;

}

//----------------------------------

public T xGet()

{

return x;

}

//----------------------------------

public U yGet()

{

return y;

}

}

1. The method below is a **generic method**. Use it in a program that calls it several times, passing it arrays of different types (Integer, Double, String, and int). Does it work for all types?

public static <T> void display(T[] a)

{  
 for (int i = 0; i < a.length; i++)

System.out.println(a[i]);

}

1. Add the clear method to MyArrayList in C16h21.java. Your clear method should work like the clear method in the ArrayList class (see Section 9.12). Add code to the main method that tests your clear method.
2. Add the indexOf(Object obj) method to MyArrayList in C16h22.java. Your indexOf method should work like the indexOf(Object obj) method in the ArrayList class (see Section 9.12). Add code to the main method that tests your indexOf method.
3. Add the remove(int index) method to MyArrayList in C16h23.java. Your remove method should work like the remove(int index) method in the ArrayList class (see Section 9.12). Add code to the main method that tests your remove method.
4. Create a generic class MyQueue that creates a FIFO (First In First Out) data structure. Use a field qal with type ArrayList within your Queue class to hold your data. Your MyQueue class should have the following methods:

public boolean isEmpty()

Returns true if the size of the the ArrayList is 0. Otherwise, it returns false.

public int size()

Returns the current size of the ArrayList.

void enqueue(T x)

Adds x to the end of the ArrayList.

T dequeue()

Removes and returns the item at index 0 in the ArrayList.

Test your class with

class C16h24

{

public static void main(String[] args)

{

MyQueue<String> q = new MyQueue<String>();

q.enqueue("hello");

q.enqueue("goodbye");

q.enqueue("last one");

while (!q.isEmpty())

{

System.out.println(q.size());

System.out.println(r.dequeue());

}

}

}

1. Add a selection sort method (see Section 9.9) to the MyArrayList in C16h25.java. Test your class with

class C16h25

{

public static void main(String[] args)

{

MyArrayList<Integer> mal = new MyArrayList<Integer>();

mal.add(3);

mal.add(1);

mal.add(2);

mal.selectionSort();

for (int i = 0; i < mal.size(); i++)

System.out.println(mal.get(i));

}

}

1. Is the following statement legal if the OneThing class is defined as in Fig. 17.2:

OneThing<double> p = new OneThing<double>(3.0);

1. Write a generic class with two members: a private instance variable whose type is parameterized and a constructor that initializes this variable to the parameter the constructor is passed.
2. Why is the following statement illegal in a generic class, where T is the type parameter:

T r = new T()

1. Compile the program below. What error message does the compiler produce? What is the problem with this program?

class Gosh<T>

{

T x;

}

//================================================

class C17h29

{

public static void main(String[] args)

{

Gosh<int> r = Gosh<int>();

}

}

1. Can a generic class with type parameter T contain fields whose types are not T? For example, would a generic class with the following fields be legal?

T x;

String y;

int z;

Create a program and compile such a class to see if it is legal.

1. Modify MyArrayList.java in C16h31.java so that you can pass an initial capacity to its constructor. Also add the set method. Your set method should work like the set method in the ArrayList class (see Section 9.12). Add code to the main method that tests your enhancement.
2. Create a generic class Yi with type parameter T in C16h32.java that contains

T x;

public Yi(T xx)

{

x = xx;

}

public void display()  
 {  
 System.out.println(x);  
 }

Test your Yi class with

class C16h32

{

public static void main(String[] args)

{

Yi<Integer> y1 = new Yi<Integer>(20);

y1.display();

Er e = new Er();

Yi<Er> y2 = new Yi<Er>(e);

y2.display();

}

}

Er is defined as follows:

class Er

{

private int n = 2;

}

What is displayed when C16h32 runs? Why is the value in y1 displayed but not the value in y2?

1. Why is line 18 in Fig. 16.9 not illegal (it is a downcast).
2. Can a generic class with type parameter T contain fields whose types are also generic classes with type parameter T? For example, is the MyStack class below legal? Use it in a program that reads in the numbers in the t1.txt file and displays them in reverse order. MyStack is in C16h34.java.

import java.util.ArrayList;

class MyStack<T>

{

private ArrayList<T> s = new ArrayList<T>();

//----------------------------------

public void push(T x)

{

s.add(x);

}

//----------------------------------

public T pop()

{

return s.remove(s.size() - 1);

}

//----------------------------------

public boolean isEmpty()

{

return s.size() == 0;

}

}

The class that is passed to the MyStack class is, in turn, passed to the ArrayList class within MyStack. Thus, if we execute

MyStack<Integer> s = new MyStack<Integer>();

the object constructed would contain an object of type ArrayList<Integer>. But if we execute

MyStack<String> s = new MyStack<String>();

then the object constructed would contain an object of type ArrayList<String>. The MyStack class implements a **stack**—a **last-in-first-out** (LIFO) data structure.

1. Same as homework problem 34 but use the predefined generic Stack class in java.util.
2. Delete all the angle brackets and the types they enclose in the main method in C17h36.java (a copy of TestOneThing.java in Fig. 16.7). Do *not* modify the OneThing class. Does the modified program compile without any errors? Does it run correctly? If you do not pass a type to a generic class, the type Object is passed by default. Add the following line at the end of main:

Integer i = p1.get();

Does it compile without error? Now try

Integer j = (Integer)p1.get();

Why does the statement above with a cast work but not the statement without the cast? If the OneThing class were not generic, but its base type were Object, in what way would it be less versatile than the original OneThing class?

1. Convert the MyLinkedList in C16h37.java to a generic class. Test your new clas

Intersession assignment

This problem set provides a good review for a final exam on the more difficult concepts in Chapters 1 to 16. It also is excellent preparation for the next course on Java programming, typically called “data structures”. If you do not continue to work with Java between courses, you will probably forget a lot. You will then start the next course without the proper preparation. But if you do at least some of the problems below between your first and second courses in Java, you will start the next course “ahead” in which case you will do well, learn more, and be rightfully proud of your accomplishment.

1. Add to C16h38.java (a copy of TestMyTree.java) a method that determines if the tree has the property below. It should return true or false accordingly.

Property: The data in each node is greater than or equal to the data in its tow subtrees. A null tree (i.e., a tree with no nodes) automatically satisfies this property.

Use recursion. *Hint*: Suppose the left and right subtrees off the root has this property. Then what test(s) must you make to determine if the entire tree has the property. What can you say about the data in the root node for a tree with this property? If the root node is removed, what must be done with its two subtrees to form a tree with the property above?

1. Eliminate recursion in the following programming:

class C16h39

{

public static void main(String[] args)

{

f(5);

}

public static void f(int x)

{

if(x >= 0)  
 {

System.out.println(x);

f(x-1); // tail recursive

}

else

System.out.println("bottom");

}

}

1. Eliminate recursion in the following program. Use the MyStack class (see homework problem 34).

class C16h40

{

public static void main(String[] args)

{

f(5);

}

public static void f(int x)

{

if(x >= 0)  
 {

f(x-1); // save current x on a stack (see homework problem 34)

System.out.println(x);

}

else

System.out.println("bottom");

}

}

1. In C16h41.java (a copy of TestMyLinkedList.java), add a method loopReverse that traverses the list in reverse order. Do not use recursion (i.e., use a loop instead). *Hint*: As you move down to the end of the list using a loop, save each pointer on a stack. Then pop and use these saved pointers to go backwards in the list. Use the MyStack class in homework problem 34 to save (with push) and retrieve (with pop) the node pointers.
2. Write a program that prompts the user to enter two integers. Your program should display their sum. If the user enters a non-integer for either the first or second integer, your program should not terminate, but prompt the user again. Use try and catch blocks.
3. Implement a stack class with push, pop, and isEmpty methods, but use an array instead of an ArrayList (see homework problem 34). Redo homework problem 41 using your new stack class.
4. A **stack** is a data structure in which adds and removals are from one side only (the “top” of the stack). The last item added is the first item removed. Thus, a stack is an example of a LIFO (last-in-first-out) structure. A **queue** (see homework problem 24) is a data structure in which adds are to one side and removals are from the other side. It works like the line in front of a movie theater—people get on the end of the line, the person at the head of the line gets served next. Thus, a queue is a FIFO (first-in-first-out) structure.

Redo homework problem 24, but use an array in place of an ArrayList. Use variables first and last to keep track of the first and last items on the queue. Use the file name C16h44.java.

1. Redo homework problem 24, but use a linked list in place of an ArrayList. Use variables first and last to keep track of the first and last items on the queue. Use the file name C16h45.java.
2. The tree traversal performed by the program in TestMyTree.java is an example of a **depth-first traversal** because each recursive call goes down to a lower node. Another type of traversal is a **breadth-first traversal**. In a breadth-first traversal, each level is traversed before the next lower level is traversed.

A good way to implement a breadth-first traversal is with a queue. The queue is initialized with the pointer to the root node. Then the following loop is executed while the queue is non-empty:

while queue non-empty

Dequeue the first pointer on the queue. Assign to p.

Display the data in the node p points to.

Enqueue the left and right pointers (unless null) in the node p points to.

Add to C16h46.java a bftraverse (breadth-first) traversal method using a queue implemented with an ArrayList (see homework problem 24).

1. Add to C16h47.java (a copy of TestMyTree.java) a method that returns the **height** of the tree. The height is the number of nodes in the longest path from the root to a leaf node (i.e., a node in which both pointers are null). Thus, a null tree has height 0. A tree with just the root node has height 1.
2. What is the advantage of the tree data structure compared with the singly-linked list data structure? *Hint*: Why it may be safter to walk a group of kids across a busy street in the formation of a tree rather than in the formation of a linked list?
3. What is the least number of nodes in the tree in TestMyTree.java if the tree has height 5? If the tree has the height 30.
4. What is the maximum number of nodes in the tree in TestMyTree.java if the tree has height 5? If the tree has the height 30?