3.1 H01\_31.java

3.2 list = {1, 3, 6, 10, 15, 19, 22, 24, 25, 25};

3.3 int counter = 0;  
for (Integer int :list){  
 if ( int <0){  
 counter++;  
 }  
}  
System.*out*.println(counter);

3.4 H01\_34.java

3.5 H01\_35.java

3.6 H01\_36.java

3.7 H01\_37.java

4.1 A FileNotFoundException is thrown and the program crashes.

4.2 Nothing will be thrown and the file will be created normally.

4.3 H01\_43.java

5.1 When an exception is thrown by a method M it is sent to the method that called M, call this method N. N either has to catch the exception or throw it. If N throws it the exception is sent to the method that called N, and the process repeats. When a method catches the exception, the method must implement an exception handler which will do something to handle the exception. In either case, the code following the point where the exception occurred is not executed.

5.2 A checked exception is one that must either be thrown by a method or caught and handled. They are called checked exceptions because the compiler will check to see if it is thrown or caught. Examples of checked exceptions include IOException (and all subclasses of IOException, which would include FileNotFoundException). Note that: Runtime Exceptions and its subclasses are not checked exceptions.

5.3 An unchecked exception is one that the compiler doesn’t check to see if it’s thrown or caught. Therefore, they don’t have to be thrown or caught. RuntimeException and all of its subclasses are unchecked exceptions.

5.4 Checked exceptions.

5.5 Because IndexOutOfBoundsException isn’t a checked exception.

5.6 No, the type declared in the catch clause can be a superclass of the class of the exception object:

try {  
 int a[] = {1, 2, 3};  
 a[5] = 0;  
} catch (Exception pExcept) {  
 *// The type pExcept is ArrayIndexOutOfBoundsException which  
 // is a subclass of Exception*}

5.7 The code in a finally clause is always executed whether an exception occurs or not. The purpose is to ensure that some resource that was acquired by the code in the try clause is released.

PrintWriter out;  
try {  
 Scanner in = new Scanner(new File("input.txt"));  
 out = new PrintWriter("output.txt");  
  
} catch (FileNotFoundException pExcept) {  
 *// Do something to handle the exception*} finally {  
 *// This code is executed whether an exception occurs or not.  
 // Makes sure that the output stream is closed* out.close();  
}

5.8 next() can throw NoSuchElementException and IllegalStateException, both of which are unchecked exceptions (they are subclasses of RuntimeException). nextInt() can throw InputMismatchException, NoSuchElementException and IllegalStateException, all of which are unchecked exceptions.

6.1 An instance method can only be called on an object (e.g., obj.method()). A class method (static method) can be called without an object as it belongs to the class itself not an instance of it (e.g., ClassName.method()).

class C {  
 public void instanceMethod() {  
  
 }  
  
 public void anotherInstanceMethod(D d) {  
 instanceMethod(); *// Calls instanceMethod on object this* d.instanceMethod(); *// Calls instanceMethod() on object d  
 staticMethod*(); *// Calls staticMethod() on no object* D.*classMethod*(); *// Calls classMethod() on class D* d.*classMethod*(); *// Calls classMethod() on object d (this is unusual)* }  
  
 public static void staticMethod() {  
 instanceMethod(); *// Illegal: a class method may not call an instance method* }  
}  
  
class D extends C {  
 *// Calls C.staticMethod() on no object* public static void classMethod() {  
 *staticMethod*();  
 }  
  
 *// Calls C.staticMethod() on no object* public void instanceMethod() {  
 *staticMethod*();  
 }  
}

6.2 The compiler will provide a default constructor that does nothing, i.e., for class C the provided constructor will be public C() {}.

6.3 (a) Within a class C, class methods cannot call instance methods of the same class. If a class method calls an instance method, it must be on an object so the call will be of the form object.instanceMethod(). A call to a static method that is also declared in C will be of the for someClassMethod().

(b) Instance methods of a class C can call both instance and class methods of C so in both cases the calls would be of the form someInstanceMethod() and someClassMethod() which are indistinguishable.

6.4 main() instantiates a C object and calls the default ctor on the object. Within the ctor, a call is made to lengthA() on s, but s has not been created so it is null. Therefore, the call to length() causes a NullPointerException which is not caught, so it will be thrown, to the JVM which will terminate the program with a stack trace.

6.5 H01\_65.java

6.6 H01\_66.java

6.7 H01\_65 cObj2 = new H01\_65(10);

6.8 (a) Illegal, mX is private.

(b) Illegal, mY is private.

(c) Illegal, A is private.

(d) a4 is assigned 200

(e) Illegal, ctors cannot be calls.

(f) a5 is assigned the value of mX of cObj1.

(g) mX of cObj1 is changed to 20.

(h) the mX instance variable of cObj2 is changed to the value of the mX instance variable cObj11.

(i) Illegal, getX() is not a class method.

(j) Illegal, setX() is not a class method.

(k) a7 is assigned the value of mY of class H01\_65.

(l) the mY class variable of class H01\_65 is changed to 20.

(m) a8 is assigned the value of the mY calss variable of class H01\_65.

(n) the mY class variable of class H01\_65 is changed to 20.

6.9 In f() the assignment to mX will change the mX instance variable of this object (the one on which the method is invoked) to 0.

In f() the assignment to mY will change the mY class variable of class H01\_65 to 0.

In g() the assignment to mX is illegal because mX is an instance variable and g() is a class method.

In g() the assignment to mY will change the value of the mY class variable of class H01\_65.

7.1

|  |
| --- |
| C |
| + CONST1: final int = 100;  - mX: int;  - mY: double;  - mZ: String; |
| + C(); <<ctor>>  + C(pX: int, pY: double, pZ: String); <<ctor>>  + getX(): int  + getY(): int  - getZ(): String  + setX(pY: int): void  + setY(pY: int): void  - setZ(pZ: int): void |

7.2 (a) A composition relationship because a Course object is composed of a Roster Object.

(b) An aggregation relationship exists because a Roster object aggregates one or more Student objects.

(c) A generalization or inheritance relationship because an UndergradStudent is a specialization (or subclass) of Student.

(d) A generalization or inheritance relationship because an GradStudent is a specialization (or subclass) of Student.

(e) There is none.

