Introduction

The 1-hour final exam will be given during the class time (12 noon to 1PM) on Friday, August 23. The exam will not be in the regular classroom, it will be in Guggenheim 224, the building just to the north of the EE1 building where we usually meet.

The exam will be closed book, no notes, no calculators.

The exam is based on the lectures and the homework. The questions will be similar to the questions on the homework with several small programming questions too.

Do your own work! Do not try to cheat on the exam.

Exam coverage

The final exam will be comprehensive, meaning that it will cover material from the entire course. Much of what we have done since the midterm is apply what we learned before then, so all the material that was listed in the midterm review sheet is still appropriate for this final exam.

Study the midterm review sheet and the "questions from class" *in addition to* this review sheet.

Inheritance

Extending a class allows you to build a new class by adding capabilities to an already defined class. The keyword extends is used to link the subclass to the superclass. For example, "public class PersonalFriend extends Person" starts the definition of a new class based on the existing class named Person.

The subclass is said to inherit all the members of the superclass. Access to the members may be limited depending on the access keyword that was used in the member definition (private, package (default), protected, or public).

Method overriding. A subclass can override a method that was defined in the superclass. In order to do this, the subclass defines a method with the same name and argument list (ie, the same signature) that returns the same value type as a method that was defined in the superclass. This allows the subclass to refine the behavior of the method to be more appropriate to the needs of the subclass. When an instance method is called at runtime, the JVM checks to see if there is a version of the method defined for the specific class of the object in use. If there is, it calls that method. If there is not, it checks to see if there is a version for the superclass. If there is, it calls that method. This continues all the way up the chain of inheritance to the "cosmic superclass", Object.

instanceof keyword. This keyword is used with a class or interface name to determine at run time if an object is of a particular type. The keyword is a logical operator giving a boolean result, and is used as follows. For an object reference o, you can test if it is an object of a particular type TestClass with the expression "if (o instanceof TestClass) {...}". If "o" is an instance of

TestClass, or any of its subclasses, the conditional expression will be true. Similarly, if TestInterface is a defined interface, you can test with "if (o instanceof TestInterface) {...}" If "o" implements the interface TestInterface, the conditional expression will be true.

Abstract classes. An abstract class is one that cannot be instantiated itself, but must be extended. In general, an abstract class contains one or more abstract methods. All of these abstract methods must be implemented by a subclass that extends the abstract superclass (or the subclass itself is also abstract). The benefit of abstract classes is that they can implement methods that are common to all subclasses, while leaving other methods for implementation by the subclasses. The keyword abstract is used to identify an abstract class or method.

See also the discussion of inheritance, interfaces, and abstract classes in the "questions from class" writeup.

Arrays

Arrays are a way to store a group of objects or primitive values and access them using an index value. An array is an object and it is created using the new operator or by providing initializers along with the declaration.

The array type is identified by giving the type of the elements it holds, followed by a pair of square brackets "[]". An array can be allocated with "new". For example

```
int[] countValues = new int[10];
```

allocates an array of integer values with 10 slots. Each of the ten slots can be used to hold an int value. The slots are accessed by using the name of the array, followed by the desired index in square brackets. The index values run from 0 to size-1. For example:

```
countValues[4]  = k;
```

sets the 5th slot of the countValues array to be the same value as the current value of integer k.

Arrays can be initialized at the same time they are allocated. In this case, Java will count the number of elements for you and set the array to the correct size automatically.

```
String[] county = new String[] {"King","Island","Snohomish","Pierce"};
```

This statement creates a new String array object and allocates space for four slots in the array. The slots are then initialized with references to the four String literals that were given.

The number of slots available in the array is provided in the length field. For example,

```
System.out.println(county.length);
```

would print the number 4.

Files and Streams

Sources and sinks. Data read or written by a program is considered to come from a source and go to a sink. Sources and sinks can be files, memory, the console, network ports, etc. At the lowest level, every source and sink deals in bytes, but there are access methods provided called streams which can impose a higher level structure on the data while it is being read and written.

Streams. Getting the data from the source to the sink is the job of a stream object. Streams are subclassed and layered to provide whatever functionality is needed. There are two basic stream classes used for raw data: InputStream and OutputStream. These are abstract classes and provide only the most basic methods. The fundamental methods read() and write() are abstract and are implemented in the subclasses.

Stream subclasses. FileInputStream is an important subclass of InputStream that specifies a file as the data source. FileOutputStream is an important subclass of OutputStream that specifies a file as the data sink. These classes are concrete (not abstract) but they don't provide all the functionality that you might want at the application level.

Layered stream subclasses. Additional functions for a stream are provided by passing the stream object to the constructor of a "decorator" class. These classes take a stream as input to their constructor, and return a stream. The new stream has access to the methods provided in the decorator class. Examples of classes used in layering for input are BufferedInputStream, ObjectInputStream, and ZipInputStream. Examples of classes used in layering for output are BufferedOutputStream, ObjectOutputStream, and ZipOutputStream.

End of stream on read. An expected end-of-stream condition should be detected by checking for -1 returned as the number of bytes read. An unexpected end-of-stream condition will result in an EOFException which should be caught or thrown. An exception should only occur if something unusual happens, not just that you've real all the way through to the end of the file.

File class. The File class is used to specify and manipulate pathnames. There are several overloaded constructors for the File class, providing flexibility in how directories and filenames are specified. The File() constructors create new pathname objects, not new files. There are numerous methods available to operate on a File object, including methods to check for existence and access permissions (eg, exists(), canRead(), canWrite()), create and delete files (eg, createNewFile(), createTempFile(), delete()) and to get directory information (eg, getParent(), list(), listFiles()).

Exceptions

Exceptions are thrown to signify that an unexpected error has been detected. The relevant information about an exception is stored in an object of a subclass of Throwable. Exceptions are not meant to be used for simple, expected situations, since they are more costly than simple checks like "if (var==null) {... do something ...}".

Errors and Exceptions. There are two primary types of Throwable objects. The class Error is for problems that are serious and that cannot be fixed by the application program. Error exceptions include LinkageError, OutOfMemoryError and StackOverflowError. Programs should not attempt to catch Errors. The Throwable subclass Exception is for problems of a less serious nature, which the program either may be able to fix, or should have prevented in the first place.

Exceptions and RuntimeExceptions. One subclass of Exception is RuntimeException. This class of problem does not have to be explicitly declared or caught. The programmer's job in this case is to make sure that these problems don't occur in the first place. RuntimeExceptions include IndexOutOfBoundsException and NullPointerException. All other subclasses of Exception must be explicitly declared or caught and dealt with. These are checked exceptions. If the compiler finds that you have called a method that might throw an Exception, but you have not announced that you will throw it or you have not caught it, the code will fail to compile. Examples of Exceptions that are not RuntimeExceptions include FileNotFoundException and DataFormatException.

A method declares that it might throw an Exception in the method header. The keyword throws is used after the parameter list, followed by the name of the Exception class. For example, a method could be declared as follows:

```
public void addFileData(String s) throws IOException {...}
```

Try/catch. The code that is using a method that might throw an exception must be prepared for the event. The calling method can either state that it may throw an exception as discussed in the previous item, or it can catch the exception when it happens. In order to do this, it needs to surround the possible problem call with a try/catch block. The syntax is

```
try {... possible problem ...}
catch (<ClassOfException> e) { ... deal with problem ...}
finally { ... executed no matter what happens, including an Exception for which there is
no catch ...}
```

Multiple catch blocks are allowed, and are searched in order from top to bottom if an Exception occurs. Optionally, an additional block can be specified using finally, which will be executed after the try block, even if an Exception occurs for which there is no catch block.

Methods defined in the Throwable class. There are useful methods defined in Throwable which are inherited by all Exception classes, primarily related to printing the message associated with an Exception, and the stack trace. These methods can be used to provide error information to the programmer for debugging purposes.

It is possible to extend the Exception class for your own specific error conditions. The subclass can just be a new name, if that is enough, or it can define added variables, methods, and constructors as appropriate.

Packages

Import and package statements. The import statement is used to tell the compiler where to find classes that you have referred to by class name only. It does not include code in the compilation, it just tells the compiler how to find the class files it needs. The package statement tells the compiler that a class is part of a particular package, and thus the class files can be found according to a particular directory structure.

Static variables and methods

You should know that an application starts executing in the "main" method of the class that is specified to the java tool. You should know that the main method is always specified as "static void main(String[] args)" and that the user command line options are passed to the program in the String array "args".

Static variables and methods are defined once when a class is loaded, and they are not repeated when an object is created. This can be useful for defining utility methods that are not associated with any one object, and also for creating a static variable that applies to all members of the class, rather than to each individual object of the class. Static methods and variables are accessed by preceding the member name by the name of the class, rather than the name of the object reference. Thus, to access a search method in the Arrays class, you could type Arrays.binarySearch(a,key), and to access the maximum Integer value possible you could type Integer.MAX_VALUE.