

Final Exam Practice Questions

1. Short Answer Questions (10 points total)

- (a) Given the following hierarchy:
- ```
class Alpha { ... }
class Beta extends Alpha { ... }
class Gamma extends Beta { ... }
```

What order are the constructors for these classes called when a Gamma object is instantiated? (1 point)

**Gamma, Beta, and lastly, Alpha.**

- (b) What class is a superclass of every class in Java? Name three methods inherited from that class? If you have an object, `obj`, how would you call those three methods? (2 points)

**Object is the superclass of every class. Some methods that are inherited from Object are `toString`, `getClass`, `hashCode`, and `equals`.**

**You simply call the method as normal: `obj.getClass()`.**

- (c) Explain what the `extends` keyword is used for (of course, without re-using the actual word *extends*!). (1 point)

`Extends` is used when you want a class to inherit member variables and methods from another class. It is used on the subclass. For example:

```
class MyClass extends MyOtherClass { ... }
```

- (d) Write the code to print out if the file, `hello.txt`, exists (assume that any necessary imports are already included, so there is no need to write them out). Then, open the file to read in and output the 1<sup>st</sup> line of text. (2 points)

```
File f = new File(hello.txt);
System.out.println(f.exists());
Scanner in = new Scanner(f);
// in is the Scanner object, so use this to call nextLine (was f in
// previous version JV, 5/18/2015)
System.out.println(in.nextLine());
```

- (e) How many classes can extend a superclass? How many superclasses can a class extend? (1 point)

**Many classes can extend a superclass, but a subclass can only have a single superclass.**

- (f) Describe two ways in which abstract classes and interfaces differ. (1 point)

- i. Abstract classes are extended, interfaces are implemented.**
- ii. Abstract classes can contain method definitions, while interfaces can only have abstract methods.**
- iii. You can implement multiple interfaces, but you can only extend a single class (abstract or not).**
- iv. Syntactically, abstract classes start with the keyword `class`, interfaces start with the keyword `interface`.**

- (g) Name 2 wrapper classes and their related primitives. (1 point)

**Integer – `int`, Boolean – `boolean`, Double – `double`**

- (h) What is autoboxing? (1 point)

**Autoboxing is the automatic conversion of a primitive type to its related wrapper class.**

2. **Definitions** – define the following terms (1 or 2 sentences for each), and show an abbreviated code example demonstrating each term (5 or 6 lines of code, with comments where appropriate). Code does not have to be complete / runnable. (10 points)

- (a) **Encapsulation** – separating the use of a class from the implementation of a class. Also, keeping member variables private and using getters/setters or other accessor/mutator methods to control access to the the member variable.

```
class BankAccount {
 private double balance;
 public double withdraw() {
 // deducts from balance only if balance is greater than 0
 }
}
```

- (b) **Inheritance** – define a new class based on an existing class. Create a class that has the methods and member variables of another class. Also called an “is-a” relationship.

```
// using the example from (a)
public class CheckingAccount extends BankAccount {
 // methods and variables will be inherited from BankAccount
}
```

- (c) **Polymorphism** – a variable of a supertype can refer to a **value** of a subtype

```
// using examples from (a) and (b)
// notice that the declared type of the variable is BankAccount, but it
// refers to an instance of the subclass, CheckingAccount
BankAccount myBankAccount = new CheckingAccount();
```

- (d) **Aggregation** – represents a part-of-a-whole relationship; that is, there is an ownership relationship between two objects (“weak has-a”). Other objects can have references to the component object, so if the containing object no longer exists, the component objects may still be present.

```
// note that the code examples in d and e were switched in a previous
// version - JV, 5/18/2015
// there can still be references to the PApplet object outside of the
// class (and most likely, there will be!).
class Tree extends BaseDrawing {
 PApplet canvas;
 Tree(PApplet canvas) {
 this.canvas = canvas;
 }
}
```

- (e) **Composition** – also represents a part-of-a-whole relationship; that is, there is an ownership relationship between two objects (“strong has-a”). However, component object is wholly owned / fully dependent on containing the object.

```
// a constructor for a class that contains another object will create a
// new one component object that's fully encapsulated
class StackOfStrings {
 private String[] elements;
 StackOfStrings() {
 this.elements = new String[10];
 }
}
```

3. **True or False** (10 points)

- (a) **True** / False – A superclass reference can refer to a subclass object.
- (b) True / **False** – A subclass reference can refer to a superclass object.
- (c) True / **False** – A NullPointerException is a checked exception that must be caught or declared.
- (d) **True** / False – To make a class immutable, all data fields must be private and final.
- (e) True / **False** – A class may only implement a single interface.
- (f) **True** / False – A regular method (not an abstract method) can be contained in an abstract class.
- (g) **True** / False – An abstract class can be used as a data type when declaring a variable
- (h) **True** / False – An abstract method must be nonstatic.
- (i) True / **False** – Not all exceptions are runtime errors.
- (j) True / **False** – The JVM searches for a method's implementation in the object's superclasses first.

4. **What's the output?** Read the program in the 1<sup>st</sup> column. Write the output of the program in the second column. (10 points)

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> public class ConConConstructor {     public static void main(String[] args) {         A a = new A(3);     } } class A extends B {     public A(int t) {         super();         System.out.println("A's constructor!");     } } class B extends C {     public D[] ds;     public B() {         ds = new D[2];         ds[0] = new D();         ds[1] = new D();         System.out.println("B's constructor!");     } } class C {     public C() {System.out.println("C's constructor!");} } class D {     public D() {System.out.println("D's constructor!");} } </pre>                                                                                                                                                       | <p>C's constructor!<br/>D's constructor!<br/>D's constructor!<br/>B's constructor!<br/>A's constructor!</p>                                                                                      |
| <pre> public class Foo {     public static void main(String[] args) {         Object o1 = new Baz();         Object o2 = new Qux();         System.out.print(o1);         System.out.print(o2);     } } class Baz extends Qux {     public String toString() {         return "Baz"; } } class Qux {     public String toString() {         return "Qux";     } } </pre>                                                                                                                                                                                                                                                                                                                                                               | <p>BazQux</p>                                                                                                                                                                                    |
| <pre> public class Division {     public static void main( String [] args ) {         Scanner in = new Scanner (System.in);         int num1 = 0, num2 = 0;         try{             System.out.println("Dividend and divisor, plz:");             num1 = in.nextInt ();             num2 = in.nextInt ();             System.out.printf ("Quotient is %s", num1/num2);         } catch (ArithmeticException e) {             System.out.println("Can't divide by 0"); }         catch (InputMismatchException e) {             System.out.println("Not an integer!");         }         catch ( Exception e ){             System.out.println(e);             System.out.println("???");         }         in.close();     } } </pre> | <p>What's the output for the following user input?</p> <p>5 and 2.5</p> <p><b>Not an integer!</b></p> <p>10 and 5</p> <p><b>Quotient is 2</b></p> <p>5 and 0</p> <p><b>Can't divide by 0</b></p> |

5. Finding and fixing bugs. (10 points)

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> public class ReverseInputBuggy {      public static void main( String [] args ) {          # should be System.in         Scanner in = new Scanner(input.in);          int numInts = in.nextInt();          # String[] strings = new String[numInts]         String strings;          # Start at 0         for (int i = 1; i &lt; numInts; i++) {              strings[i] = in.next();          }         # should start at numInts - 1         for (int i = numInts; i &gt;= 0; i--) {              System.out.println(strings[i]);          }     } } </pre>                                                                                                                                                                                                                                                                                  | <p>The code on the left asks for a number. Based on that number, it will ask for that number of additional input. It will then print out the numbers inputted in reverse order. There are 2 compilation errors, and 2 runtime errors. <b>Find</b> and <b>correct</b> them.</p> |
| <pre> public class ClassesBuggy {      public static void main(String[] args) {          MyClass3 m3 = new MyClass3();      }      abstract class MyClass1 {          public String myMethod() {return "class1";}         # abstract methods do not have a body         abstract int getNumber() {return 1;}      }      class MyClass2 {          public String myMethod() {return "class2";}      }     // can only extend one class, casing for myClass1     class MyClass3 extends MyClass2, MyClass1 {         // Override must match return type of parent         @Override         public int myMethod() {return 3;}          // if MyClass1 were actually extended, its         // abstract methods would have to be implemented         // in this class, or this class would have to be         // abstract JV, 5/18/2015      } } </pre> | <p>Find at least two errors in the code to the left (there are more than two). Write a short description of why there's an error next to the error.</p>                                                                                                                        |

6. **Multiple Choice.** There might be more than one correct answer - mark all that apply.

- (a) A subclass inherits \_\_\_\_\_ from its superclass.
- i. private methods      **ii. protected methods**      **iii. public methods**      **iv. static methods**
- (b) When you implement a method that is defined in a superclass, you \_\_\_\_\_ the original method.
- i. overload      ii. **override**      iii. call      iv. copy
- (c) An ArithmeticException is a subclass of:
- i. **Object**      ii. **Throwable**      iii. **Exception**      iv. **RuntimeException**      v. Catchable
- (d) Checked exceptions:
- i. are not checked by the compiler  
ii. must be dealt with in a try catch  
iii. must be declared  
iv. must be dealt with in a try catch and must be declared  
v. **must be either dealt with in a try catch or declared**
- (e) What is the output of the following code?
- ```
public static void main(String[] args) {  
    Object o = new Object();  
    String d = (String) o;  
    System.out.println(d);  
}
```
- i. NullPointerException ii. **ClassCastException** iii. "" iv. d
- (f) What code would be used to convert the following variable into a double?
- ```
String s = 5.0
```
- i. (double) s  
ii. s.toDouble()  
**iii. Double.parseDouble(s)**  
iv. cannot convert a String to a double
- (g) This kind of file contains your java source code:
- i. .class      ii. **.java**      iii. .txt      iv. .bytecode
- (h) If you would like to join strings together without creating new strings every time they are joined, use:
- i. Concatenation (s + s)  
**ii. StringBuilder**  
**iii. StringBuffer**  
iv. the String constructor (new String(s + s))
- (i) What is the contents of args[1] if your commandline arguments (through Eclipse Run Configuration) are: foo bar baz (that is, if you run your file as java MyClass foo bar baz)
- i. error      ii. foo      iii. **bar**      iv. baz

7. **Fill in the missing code.** Finish the implementations for the following... ( 10 points)

(a) binary search

```
public static int binarySearch(int[] numbers, int n) {
 int start = 0;
 int end = numbers.length - 1;
 while (start <= end) {
 int mid = (start + end) / 2;
 if(numbers[mid] == n) {
 return mid;
 } else if (numbers[mid] > n) {
 end = mid - 1;
 } else {
 start = mid + 1;
 }
 }
 return -1;
}
```

(b) fill in the implementations for the class and constructor below so that the output of the 1<sup>st</sup> and 2<sup>nd</sup> columns are both 42.

```
class C {
 public int foo ()
 {
 return 10;
 }
}
class S extends C {
 // finish this class definition
 int n;
 public S(int n) {
 this.n = n;
 }
 // JV, 5/18/2015
 public int foo() {
 return n;
 }
}
class Main {
 public static void main(
 String [] args) {
 S x = new S(42);
 System.out.println(x.foo());
 }
}
```

```
class C {
 private int x;
 public C(int x) {
 this.x = x;
 }
 public int foo () {
 return x;
 }
}
class S extends C {
 public S() {
 // finish this constructor
 super(42);
 }
}
class Main {
 public static void main(
 String [] args) {
 S x = new S();
 System.out.println(x.foo());
 }
}
```

8. Create a method called `decode`. It will translate a string encoded with ROT13, a simple “encryption” algorithm that shifts every letter forward 13 characters. For example, the letter `a` would be shifted to `n`. To “decode” the letter `n`, go back 12 letters to `a`. Write a method that:
- (a) takes a ROT13 encoded String as an argument
  - (b) returns the decoded version of the string
  - (c) for example: `System.out.println(decode("njrfbzr cbjre"))`; // gives back awesome power
  - (d) to decode, shift each letter back by 13
  - (e) if shift results in going back before the letter, `a`, continue from the end of the alphabet and go backwards - for example, the letter `j` would be shifted back to `w`
  - (f) only decode lowercase letters (punctuation, spaces, etc., should remain the same)

```
public static String decode(String s) {
 int shift = 13;
 char[] result = new char[s.length()];
 for(int i = 0; i < s.length(); i++) {
 char ch = s.charAt(i);
 if (ch >= 97 && ch <= 122) {
 int normalized = ch - 97;
 int shifted = normalized - shift;
 shifted = shifted < 0 ? 26 + shifted : shifted;
 result[i] = (char) (shifted + 97);
 } else {
 result[i] = ch;
 }
 }
 return new String(result);
}

// you can also use StringBuilder (and return the result of toString)
```



9. Write the following program:

- (a) Create a Rectangle class; it should have a width and a height, and it should be able to derive an area and String representation of itself based on width and height (the String representation should be: "[width]x[height], area:[area]" - 10x2, area:20.00)
- (b) Write a program that asks the user for a width and then a height 10 times
  - i. assume that the input is always valid (you don't have to take care of invalid input)
  - ii. assume that you're writing code that goes into a main method (you don't have to write the main method header... any imports are magically imported for you)
  - iii. create a rectangle object for every set of width and height entered by the user
  - iv. print out the rectangle with the largest area
- (c) Example output (everything after > is user input):

```
Width for rect 1 > 2
Height for rect 1 > 8
Width for rect 2 > 10
.... (continues through rect 10)
The largest rect is: 12x6, area:72.00
```

```
import java.util.Scanner;

public class TestRectangle {
 public static void main(String[] args) {
 Scanner in = new Scanner(System.in);
 Rectangle maxR = null;
 for(int i = 1; i < 11; i++) {
 System.out.printf("Width for rect %d > ", i);
 int w = in.nextInt();
 System.out.printf("Height for rect %d > ", i);
 int h = in.nextInt();
 Rectangle r = new Rectangle(w, h);
 if (maxR == null || r.area() > maxR.area()) {
 maxR = r;
 }
 }
 System.out.println(maxR);
 }
}

class Rectangle {
 private int w;
 private int h;
 public Rectangle(int w, int h) {
 this.w = w;
 this.h = h;
 }

 public double area() {
 return w * h;
 }

 public String toString() {
 return(String.format("%sx%s, area:%.2f", w, h, area()));
 }
}
```

10. Using the Rectangle class from the previous question, write the following program.

- (a) Read a file called rectangles.txt
- (b) Each line in the file is in the format [width]x[height]
- (c) Create and store a new Rectangle object for every line (use whatever method you like)
- (d) Print out all of the rectangles in reverse order
- (e) Print out all of the rectangles in descending order or area

```
public class TestRectangle2 {
 public static void main(String[] args) throws IOException {
 File f = new File("rectangles.txt");
 Scanner in = new Scanner(f);
 ArrayList<Rectangle> rectangles = new ArrayList<Rectangle>();
 while (in.hasNext()) {
 String line = in.nextLine();
 String[] parts = line.split("x");
 rectangles.add(new Rectangle(
 // the second parameter should come from the second
 // element of parts (it was the first in the previous version
 // - JV, 5/18/2015
 Integer.parseInt(parts[0]), Integer.parseInt(parts[1])));
 }
 for(int i = rectangles.size() - 1; i >= 0; i--) {
 System.out.println(rectangles.get(i));
 }
 }
}
```

11. Create three classes, Cat, Dog, and Animal. One of the classes must contain an abstract method. Base your class definitions on the the following sample code and its output:

```
Animal[] animals = {
 new Cat("frisky", 7), new Cat("fluffy", 8),
 new Dog("fetch", 10), new Dog("fido", 20)
};

for(Animal animal: animals) {
 System.out.printf("%s - %s\n",
 animal.getName(), animal.makeNoise());
}
System.out.println(((Cat) animals[0]).lives);
```

Running the code above prints out:

```
frisky - meow
fluffy - meow
fetch - woof
fido - woof
9
```

```
abstract class Animal {
 int weight;
 // in a previous version, this was private; it should
 // be public to ensure the other classes work JV, 5/18/2015
 public String name;

 public String getName() {
 return name;
 }
 public abstract String makeNoise();
}

class Dog extends Animal {
 public Dog(String name, int weight) {
 this.name = name;
 this.weight = weight;
 }
 public String makeNoise() {
 return "woof";
 }
}

class Cat extends Animal {
 int lives = 9;
 public Cat(String name, int weight) {
 this.name = name;
 this.weight = weight;
 }
 public String makeNoise() {
 return "meow";
 }
}
```

12. Write a CalendarDate class that has two `private` data fields day, and month, both ints.

- (a) it should have a default constructor that sets the date to a random day in May (May has 31 days)
- (b) it should have an overloaded constructor that sets the date to a the month and day passed in; if the month is not valid, then make the program crash with an `IllegalArgumentException`
- (c) an Array of CalendarDate classes should be sortable by using `Arrays.sort()`... write the code necessary to make this feature work
- (d) passing a CalendarDate Array to `Arrays.sort` will sort the Array by chronological order (for example, Jan 1 would go before Feb 1)

```
public class CalendarDate implements Comparable<CalendarDate> {
 int month;
 int date;

 public CalendarDate() {
 this.month = 5;
 this.date = (new Random()).nextInt(30) + 1;
 }

 public CalendarDate(int m, int d) {
 if(m < 1 || m > 12) {
 throw new IllegalArgumentException();
 }
 month = m;
 date = d;
 }

 public int compareTo(CalendarDate other) {
 int ret = 0;
 if (this.month == other.month && this.date == other.date) {
 ret = 0;
 } else if(this.month > other.month
 || this.month == other.month && this.date > other.date) {
 ret = 1;
 } else if(this.month < other.month
 || this.month == other.month && this.date < other.date) {
 ret = -1;
 }
 return ret;
 }
}
```

13. Using the class definition for CalendarDate in the previous question, write:

- (a) a method that tests for equality; this method should override the one that is inherited from Object – two dates should be considered equal if their month and date are the same
- (b) a method that overrides Object's toString method; it should represent a CalendarDate object as [month]/[date], for example, 5/31
- (c) a main method that:
  - i. creates an Array of 3 CalendarDate objects, 2 of them random, the third set to January 2<sup>nd</sup> explicitly
  - ii. sorts the Array (don't write your own sort for this, take advantage of the fact that an Array of CalendarDates can be sorted!)
  - iii. print out the entire sorted Array as a String
  - iv. finally, creates a new random date object – checks if it's in the Array, and prints out found if it exists in the Array

```
public static void main(String[] args) {
 CalendarDate[] dates = {
 new CalendarDate(), new CalendarDate(), new CalendarDate(1, 2)};
 Arrays.sort(dates);
 System.out.println(Arrays.toString(dates));
 CalendarDate date = new CalendarDate();
 for(CalendarDate d:dates) {
 if (date.equals(d)) {
 System.out.println("Found " + date.toString());
 }
 }
}

@Override
public boolean equals(Object obj) {
 CalendarDate other = (CalendarDate) obj;
 if (this.month == other.month && this.date == other.date) {
 return true;
 } else {
 return false;
 }
}

@Override
public String toString() {
 return String.format("%s/%s", month, date);
}
```

14. For the purpose of this question, assume that you are developing static methods for your own statistics library MyStats (it is similar to the Java's Math library that you have been using throughout the semester).
- (a) Write a public static method threeEqual() for MyStats that takes three int values as arguments and returns true if all three numbers are equal, false otherwise.
  - (b) Write a public static method median() for MyStats that takes as an argument an array of sorted integer values and returns the middle value (if there are an even number of elements, return the average of the two middle values). Your method should first verify if the array is sorted, and if it is not it should throw an IllegalArgumentException.
  - (c) Give a single Java expression that a user of the MyStats class could write to test whether the medians of three arrays of integer values int[] a, int[] b, and int[] c are all equal.

```
public class MyStats {
 public static void main(String[] args) {
 // one-liner
 MyStats.threeEqual(MyStats.median(a), MyStats.median(b),
 MyStats.median(c))
 }
 public static boolean threeEqual(int a, int b, int c) {
 return a == b && b == c;
 }

 public static int median(int[] nums) {
 if(!sorted(nums)) {
 throw new IllegalArgumentException();
 }
 int median = nums.length / 2;
 if(nums.length % 2 == 0) {
 return (nums[median] + nums[median - 1]) / 2;
 }
 return nums[median];
 }

 private static boolean sorted(int[] arr) {
 if (arr.length < 2) {
 return true;
 }
 for(int i = 1; i < arr.length; i++) {
 int prev = i - 1;
 if(arr[i] < arr[prev]) {
 return false;
 }
 }
 return true;
 }
}
```

15. Write the method `sumSeries()` that given an integer argument `n` will return as a double the following sum:

$$1/n + 2/(n-1) + 3/(n-2) + \dots + (n-1)/2 + n/1$$

Do not write a full program with input and output, just the method.

Hints/Notes:

- (a) You need only a single loop.
- (b) The return value is a double, make sure that as the sum is computed using double division and not integer division.
- (c) You don't need to check for valid values of `n`, assume it's an integer  $> 0$ .

```
public class Series {
 public static void main(String[] args) {
 System.out.println(sumSeries(3));
 }
 public static double sumSeries(int n) {
 double total = 0;
 int numerator = 1;
 for(int i = n; i >= 1; i --) {
 total += numerator / (float) i;
 numerator += 1;
 }
 return total;
 }
}
```

| Char  | Dec | Char | Dec | Char | Dec | Char  | Dec |
|-------|-----|------|-----|------|-----|-------|-----|
| (nul) | 0   | (sp) | 32  | @    | 64  | `     | 96  |
| (soh) | 1   | !    | 33  | A    | 65  | a     | 97  |
| (stx) | 2   | "    | 34  | B    | 66  | b     | 98  |
| (etx) | 3   | #    | 35  | C    | 67  | c     | 99  |
| (eot) | 4   | \$   | 36  | D    | 68  | d     | 100 |
| (eng) | 5   | %    | 37  | E    | 69  | e     | 101 |
| (ack) | 6   | &    | 38  | F    | 70  | f     | 102 |
| (bel) | 7   | '    | 39  | G    | 71  | g     | 103 |
| (bs)  | 8   | (    | 40  | H    | 72  | h     | 104 |
| (ht)  | 9   | )    | 41  | I    | 73  | i     | 105 |
| (nl)  | 10  | *    | 42  | J    | 74  | j     | 106 |
| (vt)  | 11  | +    | 43  | K    | 75  | k     | 107 |
| (np)  | 12  | ,    | 44  | L    | 76  | l     | 108 |
| (cr)  | 13  | -    | 45  | M    | 77  | m     | 109 |
| (so)  | 14  | .    | 46  | N    | 78  | n     | 110 |
| (si)  | 15  | /    | 47  | O    | 79  | o     | 111 |
| (dle) | 16  | 0    | 48  | P    | 80  | p     | 112 |
| (dcl) | 17  | 1    | 49  | Q    | 81  | q     | 113 |
| (dc2) | 18  | 2    | 50  | R    | 82  | r     | 114 |
| (dc3) | 19  | 3    | 51  | S    | 83  | s     | 115 |
| (dc4) | 20  | 4    | 52  | T    | 84  | t     | 116 |
| (nak) | 21  | 5    | 53  | U    | 85  | u     | 117 |
| (syn) | 22  | 6    | 54  | V    | 86  | v     | 118 |
| (etb) | 23  | 7    | 55  | W    | 87  | w     | 119 |
| (can) | 24  | 8    | 56  | X    | 88  | x     | 120 |
| (em)  | 25  | 9    | 57  | Y    | 89  | y     | 121 |
| (sub) | 26  | :    | 58  | Z    | 90  | z     | 122 |
| (esc) | 27  | ;    | 59  | [    | 91  | {     | 123 |
| (fs)  | 28  | <    | 60  | \    | 92  | }     | 124 |
| (gs)  | 29  | =    | 61  | ]    | 93  | }     | 125 |
| (rs)  | 30  | >    | 62  | ^    | 94  | ~     | 126 |
| (us)  | 31  | ?    | 63  | _    | 95  | (del) | 127 |