Primitive vs. Reference Variables

```java
int x;
boolean b;
x = 17;
b = false;
```
Primitive vs. Reference Variables

```java
int x;
boolean b;
x = 17;
b = false;
```
Primitive vs. Reference Variables

```java
int x;
boolean b;
x = 17;
b = false;
```
Primitive vs. Reference Variables

```java
int x;
boolean b;
x = 17;
b = false;
```
Primitive vs. Reference Variables

```java
int x;
boolean b;
x = 17;
b = false;

String s;
s = "Hello";
```
Primitive vs. Reference Variables

```java
int x;
boolean b;
x = 17;
b = false;

String s;
s = "Hello";
```
Primitive vs. Reference Variables

int x;
boolean b;
x = 17;
b = false;

String s;
s = “Hello”;
Reference Variables

What happens in memory when the following code is executed given the definition of the Foo class on the side.

Foo f1;
Foo f2;
f1 = new Foo (5);
f2 = f1;
f1.set(17);
System.out.println( f1.get() );
System.out.println( f2.get() );
Reference Variables

What happens in memory when the following code is executed given the definition of the Foo class on the side.

Foo f1;
Foo f2;
f1 = new Foo (5);
f2 = f1;
f1.set(17);
System.out.println( f1.get() );
System.out.println( f2.get() );
Reference Variables

What happens in memory when the following code is executed given the definition of the `Foo` class on the side.

```java
class Foo {
    private int n;
    public Foo ( int n ) {
        this.n = n;
    }
    public void set( int n) {
        this.n = n;
    }
    public int get( ) {
        return n;
    }
}
```

```java
Foo f1;
Foo f2;
f1 = new Foo (5);
f2 = f1;
f1.set(17);
System.out.println( f1.get() );
System.out.println( f2.get() );
```
Reference Variables

What happens in memory when the following code is executed given the definition of the `Foo` class on the side.

```java
public class Foo {
    private int n;
    public Foo ( int n ) {
        this.n = n;
    }
    public void set( int n) {
        this.n = n;
    }
    public int get( ) {
        return n;
    }
}
```

```java
Foo f1;
Foo f2;
f1 = new Foo (5);
f2 = f1;
f1.set(17);
System.out.println( f1.get() );
System.out.println( f2.get() );
```
Reference Variables

What happens in memory when the following code is executed given the definition of the `Foo` class on the side.

```java
public class Foo {
    private int n;
    public Foo ( int n ) { this.n = n; }
    public void set( int n) { this.n = n; }
    public int get( ) { return n; }
}
```

Foo f1;
Foo f2;
f1 = new Foo (5);
f2 = f1;
f1.set(17);
System.out.println( f1.get() );
System.out.println( f2.get() );
Reference Variables

What happens in memory when the following code is executed given the definition of the `Foo` class on the side.

```java
public class Foo {
    private int n;
    public Foo ( int n ) {
        this.n = n;
    }
    public void set ( int n ) {
        this.n = n;
    }
    public int get () {
        return n;
    }
}
```

Foo f1;
Foo f2;
f1 = new Foo (5);
f2 = f1;
f1.set(17);
System.out.println( f1.get() );
System.out.println( f2.get() );

Output: 17
17
Formatted Console Output

System.out.printf( )

Syntax:
System.out.printf( format, item1, item2, ..., itemN);

format is a string that consists of substrings and format specifiers.

item1...itemN are values (numeric, character, string, Boolean), either variables or constants
Formatted Console Output
System.out.printf( )

Simple format specifier consists of a percent sign, %, followed by a letter:
   b, c, d, f, e, s, h or n
that indicates the type of the item corresponding to the format specifier.

%b  a Boolean value
%c  a character
%d  a decimal integer
%f  a floating point number
%e  a number in scientific notation
%s  a string
%h  a hashcode (we will see what hashcodes are later)
%n  a newline (platform specific newline character- use %n instead of \n for greater compatibility)
System.out.printf("Area of circle with radius %f is %f.\n",
                   radius, area);
System.out.printf("Given that x is %b and y is %b,\n" +
                   " the value of x && y is %b.\n", x, y, x&&y);
System.out.printf("Student: %s, tuition: %f.\n",
                   name, tuition);

Output:
Area of circle with radius 5.250000 is 86.587594.
Given that x is true and y is false, the value of x && y is false.
Student: John Smith, tuition: $25036.210000.
Formatted Console Output
System.out.printf( )

Example:
System.out.printf("Area of circle with radius %f is %f.",
       radius, area);
System.out.printf("Given that x is %b and y is %b," +
       " the value of x && y is %b.", x, y, x&&y );
System.out.printf("Student: %s, tuition: %f.",
       name, tuition );

Output:
Area of circle with radius 5.250000 is 86.587594.
Given that x is true and y is false, the value of x && y is false.
Student: John Smith, tuition: $25036.210000.
System.out.printf("Area of circle with radius %f is %f.",
    radius, area);
System.out.printf("Given that x is %b and y is %b,
    the value of x && y is %b.", x, y, x&&y );
System.out.printf("Student: %s, tuition: %f.",
    name, tuition );

Output:
Area of circle with radius 5.250000 is 86.587594.
Given that x is true and y is false, the value of x && y is false.
Student: John Smith, tuition: $25036.210000.
Formatted Console Output

System.out.printf( )

Format specifier can include optional flags, field width and precision in addition to the conversion code (indicating the type of the value):

% [flags] [width] [.precision] conversion-code

Flags
- left-justify ( default is to right-justify )
+ output a plus ( + ) or minus ( - ) sign for a numerical value
0 forces numerical values to be zero-padded ( default is blank padding )
, comma grouping separator (for numbers > 1000)
  space will display a minus sign if the number is negative or a space if it is positive

Width Specifies the field width for outputting the argument and represents the minimum number of characters to be written to the output. Include space for expected commas and a decimal point in the determination of the width for numerical values.

Precision Used to restrict the output depending on the conversion. It specifies the number of digits of precision when outputting floating-point values or the length of a substring to extract from a String. Numbers are rounded to the specified precision.
Example: Formatting Real Numbers

double d1 = 3294923.341203830, d2 = -32.1;

//display dollar values:
// with exactly two decimal places,
// aligned at the decimal point
// with commas separating every three digits

System.out.printf("Dollar ammounts: %n +
   \t\t$%,12.2f%n +
   \t\t$%,12.2f%n", d1, d2);
Example:
Formatting Integers

int n1 = 47, n2 = -7432;

//print numbers aligned right within the 10 spaces field
// (use commas to separate every three digits and display the sign
// for each number)
System.out.printf("Numbers: %n +
    ",+10d after%n" +
    ",+10d after%n", n1, n2);

//print numbers aligned left within the 10 spaces field
// (use commas to separate every three digits)
System.out.printf("Numbers: %n +
    ",-10d after%n" +
    ",-10d after%n", n1, n2);
Example:
Formatting Strings

String s1 = "Hello Earthlings";
String s2 = "This is Donald Duck talking";

//display text using field of at least 40 characters, aligned right
System.out.printf("String 1: %40s000%n" +
                   "String 2: %40s000%n", s1,s2);

//display text using field of at least 40 characters, aligned left
System.out.printf("String 1: %-40s000%n" +
                   "String 2: %-40s000%n", s1,s2);

//display text using field of at least 40 characters, aligned left,
//but display on the first 15 characters of each string
System.out.printf("String 1: %-40.15s000%n" +
                   "String 2: %-40.15s000%n", s1,s2);
public static String format(String format, Object... args)

Returns a formatted string using the specified format string and arguments.

The locale always used is the one returned by Locale.getDefault().

Parameters:
format - A format string
args - Arguments referenced by the format specifiers in the format string. If there are more arguments than format specifiers, the extra arguments are ignored. The number of arguments is variable and may be zero. The maximum number of arguments is limited by the maximum dimension of a Java array as defined by The Java™ Virtual Machine Specification. The behaviour on a null argument depends on the conversion.

Returns:
A formatted string

Throws:
IllegalFormatException - If a format string contains an illegal syntax, a format specifier that is incompatible with the given arguments, insufficient arguments given the format string, or other illegal conditions. For specification of all possible formatting errors, see the Details section of the formatter class specification.

NullPointerException - If the format is null