1. Which of the following statements are true?

Select the two correct answers.

(a) In Java, the **extends** clause is used to specify the inheritance relationship.

(b) The subclass of a non-abstract class can be declared as **abstract**.

(c) All members of the superclass are inherited by the subclass.

(d) A **final** class can be **abstract**.

(e) A class in which all the members are declared **private** cannot be declared as

**public**.

*(a) and (b)*

The extends clause is used to specify that a class extends another class. A

subclass can be declared as abstract regardless of whether the superclass was

declared as abstract. Private, overridden, and hidden members from the

superclass are not inherited by the subclass. A class cannot be declared as both

abstract and final, since an abstract class needs to be extended to be

useful, and a final class cannot be extended. The accessibility of the class is not

limited by the accessibility of its members. A class with all the members declared private can still be declared as public.

1. Which of the following statements are true?

Select the two correct answers.

(a) A class can be extended by only one class.

(b) Every Java object has a public method named equals.

(c) Every Java object has a public method named length.

(d) A class can extend any number of classes.

(e) A non-final class can be extended by any number of classes.

*(b) and (e)*

The Object class has a public method named equals, but it does not have

any method named length. Since all classes are subclasses of the Object class,

they all inherit the equals() method. Thus, all Java objects have a public

method named equals. In Java, a class can extend only a single superclass, but

there is no limit on how many subclasses can extend a superclass.

1. Given the following classes and declarations, which statements are true?

// Classes

class Foo {

private int i;

public void f() { /\* … \*/ }

public void g() { /\* … \*/ }

}

class Bar extends Foo {

public int j;

public void g() { /\* … \*/ }

}

// Declarations:

Foo a = new Bar();

Bar b = new Bar();

Select the three correct answers.

(a) The Bar class is a subclass of Foo.

(b) The statement b.f(); is legal.

(c) The statement a.j = 5; is legal.

(d) The statement a.g(); is legal.

(e) The statement b.i = 3; is legal.

*(a), (b), and (d)*

Bar is a subclass of Foo that overrides the method g(). The statement a.j = 5

is not legal, since the member j in the class Bar cannot be accessed through a Foo reference. The statement b.i = 3 is not legal either, since the private member

i cannot be accessed from outside of the class Foo.

1. Given classes A, B, and C, where B extends A, and C extends B, and where all classes implement the instance method void doIt(), how can the doIt() method in A be called from an instance method in C?

Select the one correct answer.

(a) doIt();

(b) super.doIt();

(c) super.super.doIt();

(d) this.super.doIt();

(e) A.this.doIt();

(f) ((A) this).doIt();

(g) It is not possible.

*(g)*

**It is not possible** to invoke the doIt() method in A from an instance method in

class C. The method in C needs to call a method in a superclass two levels up in the inheritance hierarchy. The super.super.doIt() strategy will not work, since

super is a keyword and cannot be used as an ordinary reference, nor can it be

accessed like a field. If the member to be accessed had been a field, the solution

would be to cast the this reference to the class of the field and use the resulting

reference to access the field. Field access is determined by the declared type of the reference, whereas the instance method to execute is determined by the actual type of the object denoted by the reference at runtime.

1. What would be the result of compiling and running the following program?

public class UserClass {

public static void main(String[] args) {

B b = new C();

System.out.println(b.max(13, 29));

}

}

class A {

int max(int x, int y) { if (x>y) return x; else return y; }

}

class B extends A {

int max(int x, int y) { return super.max(y, x) - 10; }

}

class C extends B {

int max(int x, int y) { return super.max(x + 10, y + 10); }

}

Select the one correct answer.

(a) The code will fail to compile.

(b) The code will compile, but throw an exception at runtime.

(c) The code will compile, and print 13 at runtime.

(d) The code will compile, and print 23 at runtime.

(e) The code will compile, and print 29 at runtime.

(f) The code will compile, and print 39 at runtime.

*(e)*

The code will compile without errors. None of the calls to a max() method are

ambiguous. When the program is run, the main() method will call the max()

method on the C object referred to by the reference b with the parameters 13 and 29. This method will call the max() method in B with the parameters 23 and 39. The max() method in B will in turn call the max() method in A with the

parameters 39 and 23. The max() method in A will return 39 to the max()

method in B. The max() method in B will return 29 to the max() method in C.

The max() method in C will return 29 to the main() method.

1. Which is the simplest expression that can be inserted at (1), so that the program prints the value of the text field from the Message class?

// File: MyClass.java

class Message {

// The message that should be printed:

String text = “Hello, world!”;

}

class MySuperclass {

Message msg = new Message();

}

public class MyClass extends MySuperclass {

public static void main(String[] args) {

MyClass object = new MyClass();

object.print();

}

public void print() {

System.out.println(/\*(1)insert the simplest expression \*/);

}

}

Select the one correct answer.

(a) text

(b) Message.text

(c) msg.text

(d) this.msg.text

(e) super.msg.text

(f) this.super.msg.text

*(c)*

The simplest way to print the message in the class Message would be to use

msg.text. The main() method creates an instance of MyClass, which results

in the creation of a Message instance. The field msg denotes this Message

object in MySuperclass and is inherited by the MyClass object, as this field

has default accessibility. Thus, the message in the Message object can be accessed directly by msg.text in the print() method of MyClass, and also by

this.msg.text and super.msg.text.

1. What would be the result of compiling and running the following program?

class Vehicle {

static public String getModelName() { return “Volvo”; }

public long getRegNo() { return 12345; }

}

class Car extends Vehicle {

static public String getModelName() { return “Toyota”; }

public long getRegNo() { return 54321; }

}

public class TakeARide {

public static void main(String[] args) {

Car c = new Car();

Vehicle v = c;

System.out.println(“|” + v.getModelName() + “|” + c.getModelName() + ”|” + v.getRegNo() + “|” +

c.getRegNo() + “|”);

}

}

Select the one correct answer.

(a) The code will fail to compile.

(b) The code will compile, and print |Toyota|Volvo|12345|54321| at runtime.

(c) The code will compile, and print |Volvo|Toyota|12345|54321| at runtime.

(d) The code will compile, and print |Toyota|Toyota|12345|12345| at runtime.

(e) The code will compile, and print |Volvo|Volvo|12345|54321| at runtime.

(f) The code will compile, and print |Toyota|Toyota|12345|12345| at runtime.

(g) The code will compile, and print |Volvo|Toyota|54321|54321| at runtime.

*(g)*

In the class Car, the static method getModelName() hides the static method of

the same name in the superclass Vehicle. In the class Car, the instance method

getRegNo() overrides the instance method of the same name in the superclass

Vehicle. The declared type of the reference determines the method to execute

when a static method is called, but the actual type of the object at runtime

determines the method to execute when an overridden method is called.

1. Which constructors can be inserted at (1) in MySub without causing a compile time error?

class MySuper {

int number;

MySuper(int i) { number = i; }

}

class MySub extends MySuper {

int count;

MySub(int count, int num) {

super(num);

this.count = count;

}

// (1) INSERT CONSTRUCTOR HERE

}

Select the one correct answer.

(a) MySub() {}

(b) MySub(int count) { this.count = count; }

(c) MySub(int count) { super(); this.count = count; }

(d) MySub(int count) { this.count = count; super(count);}

(e) MySub(int count) { this(count, count); }

(f) MySub(int count) { super(count); this(count, 0); }

*(e)*

The class MySuper does not have a no-argument constructor. This means that

constructors in subclasses must explicitly call the superclass constructor and

provide the required parameters. The supplied constructor accomplishes this by

calling super(num) in its first statement. Additional constructors can accomplish

this either by calling the superclass constructor directly using the super() call, or

by calling another constructor in the same class using the this() call, which in

turn calls the superclass constructor. (a) and (b) are not valid, since they do not call the superclass constructor explicitly. (d) fails, since the super() call must always be the first statement in the constructor body. (f) fails, since the super() and this() calls cannot be combined.

1. Which of the following statements is true?

Select the one correct answer.

(a) A super() or this() call must always be provided explicitly as the first

statement in the body of a constructor.

(b) If both a subclass and its superclass do not have any declared constructors, the implicit default constructor of the subclass will call super() when run.

(c) If neither super() nor this() is specified as the first statement in the body

of a constructor, this() will implicitly be inserted as the first statement.

(d) If super() is the first statement in the body of a constructor, this() can be

declared as the second statement.

(e) Calling super() as the first statement in the body of a constructor of a

subclass will always work, since all superclasses have a default constructor.

*(b)*

In a subclass without any declared constructors, the default constructor will call

super(). The use of the super() and this() statements are not mandatory as

long as the superclass has a default constructor. If neither super() nor this()

is declared as the first statement in the body of a constructor, then the default

super() will implicitly be the first statement. A constructor body cannot have

both a super() and a this() statement. Calling super() will not always

work, since a superclass might not have a default constructor.

1. What will the following program print when run?

public class MyClass {

public static void main(String[] args) {

B b = new B(“Test”);

}

}

class A {

A() { this(“1”, “2”); }

A(String s, String t) { this(s + t); }

A(String s) { System.out.println(s); }

}

class B extends A {

B(String s) { System.out.println(s); }

B(String s, String t) { this(t + s + “3”); }

B() { super(“4”); };

}

Select the one correct answer.

(a) It will just print Test.

(b) It will print Test followed by Test.

(c) It will print 123 followed by Test.

(d) It will print 12 followed by Test.

(e) It will print 4 followed by Test.

*(d)*

The program will print 12 followed by Test. When the main() method is

executed, it will create a new instance of B by passing "Test" as an argument.

This results in a call to the constructor of B, which has one String parameter. The constructor does not explicitly call any superclass constructor or any overloaded constructor in B using a this() call; instead, the no-argument constructor of the superclass A is called implicitly. The no-argument constructor of A calls the constructor in A that has two String parameters, passing it the argument list ("1", "2"). This constructor calls the constructor with one String parameter, passing the argument "12". This constructor prints the argument, after implicitly invoking the no-argument constructor of the superclass Object. Now the execution of all the constructors in A is completed, and execution continues in the constructor of B. This constructor now prints the original argument "Test" and returns to the main() method.

1. Which statement about the following program is true?

public class MyClass {

public static void main(String[] args) {

A[] arrA;

B[] arrB;

arrA = new A[10];

arrB = new B[20];

arrA = arrB; // (1)

arrB = (B[]) arrA; // (2)

arrA = new A[10];

arrB = (B[]) arrA; // (3)

}

}

class A {}

class B extends A {}

Select the one correct answer.

(a) The program will fail to compile because of the assignment at (1).

(b) When run, the program will throw a java.lang.ClassCastException

in the assignment at (2).

(c) When run, the program will throw a java.lang.ClassCastException

in the assignment at (3).

(d) The program will compile and run without errors, even if the cast operator

(B[]) in the statements at (2) and (3) is removed.

(e) The program will compile and run without errors, but will not do so if the cast operator (B[]) in statements at (2) and (3) is removed.

*(c)*

The program will throw a java.lang.ClassCastException in the

assignment at (3) at runtime. The statement at (1) will compile, since the

assignment is done from a subclass reference to a superclass reference. The cast at (2) assures the compiler that arrA refers to an object that can be cast to type B[]. This will work when run, since arrA will refer to an object of type B[]. The cast at (3) assures the compiler that arrA refers to an object that can be cast to type B[]. This will not work when run, since arrA will refer to an object of type A[].

1. Which statements will cause a compile-time error in the following code?

public class MyClass {

public static void main(String[] args) {

MyClass a;

MySubclass b;

a = new MyClass(); // (1)

b = new MySubclass(); // (2)

a = b; // (3)

b = a; // (4)

a = new MySubclass(); // (5)

b = new MyClass(); // (6)

}

}

class MySubclass extends MyClass {}

Select the two correct answers.

(a) (1)

(b) (2)

(c) (3)

(d) (4)

(e) (5)

(f) (6)

*(d) and (f)*

(4) and (6) will cause a compile-time error, since an attempt is made to assign a

reference value of a supertype object to a reference of a subtype. The type of the

source reference value is MyClass and the type of the destination reference is

MySubclass. (1) and (2) will compile, since the reference is assigned a reference

value of the same type. (3) will also compile, since the reference is assigned a

reference value of a subtype.

1. Given the following class and reference declarations, what can be said about the statement y = (Sub) x?

// Class declarations:

class Super {}

class Sub extends Super {}

// Reference declarations:

Super x = null;

Sub y = null;

Select the one correct answer.

(a) It is illegal at compile time.

(b) It is legal at compile time, but might be illegal at runtime.

(c) It is definitely legal at runtime, but the cast operator (Sub) is not strictly

needed.

(d) It is definitely legal at runtime, and the cast operator (Sub) is needed.

*(b)*

The compiler will allow the statement, as the cast is from the supertype (Super) to the subtype (Sub). However, if at runtime the reference x does not denote an object of the type Sub, a ClassCastException will be thrown.

1. Given three classes A, B, and C, where B is a subclass of A, and C is a subclass of B, which one of these boolean expressions is true only when the reference o refers to an object of class B, and not to an object of class A or class C?

Select the one correct answer.

(a) (o instanceof B) && (!(o instanceof A))

(b) (o instanceof B) && (!(o instanceof C))

(c) !((o instanceof A) || (o instanceof B))

(d) (o instanceof B)

(e) (o instanceof B) && !((o instanceof A) || (o instanceof C))

*(b)*

The expression (o instanceof B) will return true if the object referred to

by o is of type B or a subtype of B. The expression (!(o instanceof C))

will return true unless the object referred to by o is of type C or a subtype of C.

Thus, the expression (o instanceof B) && (!(o instanceof C))

will return true only if the object is of type B or a subtype of B that is not C or a

subtype of C. Given objects of the classes A, B, and C, this expression will return true only for objects of class B.

1. What is the result of compiling and running the following program?

class YingYang {

void yingyang(Integer i) {

System.out.println(“Integer: ” + i);

}

void yingyang(Integer[] ints) {

System.out.println(“Integer[]: ” + ints[0]);

}

void yingyang(Integer… ints) {

System.out.println(“Integer…: ” + ints[0]);

}

}

public class RQ800A50 {

public static void main(String[] args) {

YingYang yy = new YingYang();

yy.yingyang(10);

yy.yingyang(10,12);

yy.yingyang(new Integer[] {10, 20});

yy.yingyang(new Integer(10), new Integer(20));

}

}

Select the one correct answer.

(a) The program will not compile because of errors.

(b) The program will compile, but throw an exception at runtime.

(c) The program will compile and print:

Integer: 10

Integer…: 10

Integer…: 10

Integer…: 10

(d) The program will compile and print:

Integer: 10

Integer…: 10

Integer[]: 10

Integer…: 10

*(a)*

The signatures yingyang(Integer[]) and yingyang(Integer…) are

equivalent and, therefore, are not permitted in the same class.

1. What will be the result of compiling and running the following program?

public class RQ800A20 {

static void compute(int… ia) { // (1)

System.out.print(“|”);

for(int i : ia) {

System.out.print(i + “|”);

}

System.out.println();

}

static void compute(int[] ia1, int… ia2) { // (2)

compute(ia1);

compute(ia2);

}

static void compute(int[] ia1, int[]… ia2d) { // (3)

for(int[] ia : ia2d) {

compute(ia);

}

}

public static void main(String[] args) {

compute(new int[] {10, 11}, new int[] {12, 13, 14}); // (4)

compute(15, 16); // (5)

compute(new int[] {17, 18}, new int[][] {{19}, {20}}); //(6)

compute(null, new int[][] {{21}, {22}}); // (7)

}

}

Select the one correct answer.

(a) The program does not compile because of errors in one or more calls to the

compute() method.

(b) The program compiles, but throws a NullPointerException when run.

(c) The program compiles and prints:

|10|11|

|12|13|14|

|15|16|

|19|

|20|

|21|

|22|

(d) The program compiles and prints:

|12|13|14|

|15|16|

|10|11|

|19|

|20|

|21|

|22|

*(c)*

The calls to the compute() method in the method declarations at (2) and at (3)

are to the compute() method declaration at (1), as the argument is always an

int[]. The method call at (4) calls the method at (2). The signature of the call at (4) is compute(int[], int[]) which matches the signature of the method at (2). No implicit array is created. The method call in (5) calls the method at (1). An implicit array of int is created to store the argument values.

The method calls in (6) and (7) call the method in (3). Note the type of the variable arity parameter in (3): an int[][]. The signature of the calls at (6) and (7) is compute(int[], int[][]) which matches the signature of the method at (3). No implicit array is created.