

# **Java™ Programmer Certification Mock Exam**

Dan Chisholm  
Livermore, CA 94550-4008

# **Java™ Programmer Certification Mock Exam**

by Dan Chisholm

## **Copyright © 2004, Dan Chisholm**

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means without written permission from the author and publisher, except for the inclusion of brief quotations in a review.

Published by Dan Chisholm, 849 E. Stanley Blvd. #436, Livermore, CA 94550-4008.

*ISBN: 0-9745862-0-X*

*Printing History:* June 2004, First Edition.

## **Trademarks**

All terms that are known to be trademarks or service marks have been appropriately capitalized. Java and Sun are trademarks of Sun Microsystems.

## **Warning and Disclaimer**

Every effort has been made to make this book as accurate as possible, but no warranty of any kind is expressed or implied. The author and publisher shall not be liable for any damages in connection with or arising out of the use of the information contained in this book.

## **Acknowledgements**

Many people have contributed suggestions for improvements to the content of this book, but no one has contributed more to the technical review process than Marlene Miller. Thank you Marlene for your amazing work!

I would also like to thank the hundreds of people at the JavaRanch (<http://www.javaranch.com/>) for all of the feedback that has contributed to the quality of my Web site (<http://www.danchisholm.net/>) and this book.



# Table of Contents

---

Chapter	12	<b>Method Overloading</b>	1
---------	----	---------------------------	---

---

Appendix



# Chapter 12

## Method Overloading

---

This chapter covers method overloading. A method name is overloaded if two or more methods in the same class share the same name but have different signatures. The method signature includes the name of the method and the number and types of the formal parameters. The methods could be declared in the same class. Some or all of the methods could be inherited from a superclass.

A common mistake is to confuse the overloading of method names with the overriding of instance methods or the hiding of class methods. When you see a method in a subclass that has the same name as a method in a superclass, then look at the list of parameters to determine if the subclass method overloads or overrides or hides the superclass method. If the signature of an instance method of a subclass matches the signature of an instance method of a superclass, then the subclass method overrides the superclass method. Similarly, if the signature of a class method of a subclass matches the signature of a class method of a superclass, then the subclass method hides the superclass method. If the signatures are different, then the method name is overloaded.

If a method name is overloaded and the method is invoked, which overloaded form is selected? The answer to that question depends on the answers to the following questions.

- How does the compiler determine which methods are applicable?
- Is one method declaration more specific than another?
- How do the argument types and parameter types determine which overloaded form will be selected?

How does the compiler determine which methods are applicable? The following is a simplified explanation. Class A contains four methods that overload the method name `m1`.

```
class A {
    void m1(int param1) {}
    void m1(int param1, double param2) {}
    void m1(float param1) {}
    void m1(double param1) {}
    public static void main(String[] args) {
        A a1 = new A();
        long argument1 = 1;
        a1.m1(argument1);
    }
}
```

The `main` method contains the method invocation expression `a1.m1(argument1)`. The compiler determines which implementations of `m1` have the same number of parameters as there are arguments in the

## 2 Chapter 12: Method Overloading

method invocation expression. The three methods that have the right number of parameters are `m1(int param1)`, `m1(float param1)` and `m1(double param1)`. Next, the compiler compares the argument type in the method invocation expression, `long`, to the parameter types in the method declarations. There is no exact match, but the argument of type `long` can be converted to either type `float` or type `double`; so the methods `m1(float param1)` and `m1(double param1)` are applicable.

Since two methods are applicable to the method invocation `a1.m1(argument1)`, the compiler must determine which of the two methods is the most *specific*. Suppose that two methods overload the same method name. The first method is more specific than the second if the first is able to handle any invocation that could be passed on to the second without a compile-time error. Consider the following two class declarations.

```
class B {
    void m1(Object obj) {}
    void m2(String str) {}
}
class C extends B {
    void m1(String str){super.m1(str);} // more specific than B.m1
    void m2(Object obj){super.m2(obj);} // compile-time error
}
```

The method name `m1` is overloaded. Method `C.m1` has one parameter of type `String` that can be passed on to method `B.m1` without a compile-time error; so `C.m1` is more specific than `B.m1`. The method name `m2` is also overloaded. Method `C.m2` has one parameter of type `Object` that can not be passed on to method `B.m2` without a compile-time error; so `C.m2` is not more specific than `B.m2`.

Going back to our earlier program, we can see that method `m1(float param1)` is more specific than `m1(double param1)`, because any invocation of the first can be passed on to the second without a compile-time error.

When the argument is a reference variable, it is important to remember that the argument type is determined at compile-time by the type of the reference variable. The argument type is not determined at run-time by the type of the referenced object.

# Exam 1

## Chapter 12: Method Overloading

---

### Question 1

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D extends C {
    void m1(D d) {System.out.print("D");}
    public static void main(String[] args) {
        A a1 = new A(); B b1 = new B(); C c1 = new C(); D d1 = new D();
        d1.m1(a1); d1.m1(b1); d1.m1(c1);
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: AAA
- b. Prints: ABC
- c. Prints: DDD
- d. Prints: ABCD
- e. Compile-time error
- f. Run-time error
- g. None of the above

### Question 2

```
class GFC215 {
    static String m(float i) {return "float";}
    static String m(double i) {return "double";}
    public static void main (String[] args) {
        int a1 = 1; long b1 = 2; System.out.print(m(a1)+", "+ m(b1));
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: float,float
- b. Prints: float,double
- c. Prints: double,float
- d. Prints: double,double
- e. Compile-time error
- f. Run-time error
- g. None of the above

### Question 3

```
class A {} class B extends A {} class C extends B {}
class D {
    void m1(A a) {System.out.print("A");}
    void m1(B b) {System.out.print("B");}
    void m1(C c) {System.out.print("C");}
    public static void main(String[] args) {
        A c1 = new C(); B c2 = new C(); C c3 = new C(); D d1 = new D();
        d1.m1(c1); d1.m1(c2); d1.m1(c3);
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: AAA
- b. Prints: ABC
- c. Prints: CCC
- d. Compile-time error
- e. Run-time error
- f. None of the above

## 4 Chapter 12: Method Overloading, Exam 1

### Question 4

```
class GFC216 {
    static String m(float i) {return "float";}
    static String m(double i) {return "double";}
    public static void main (String[] args) {
        char a1 = 1; long b1 = 2; System.out.print(m(a1)+", "+ m(b1));
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: float,float
- b. Prints: float,double
- c. Prints: double,float
- d. Prints: double,double
- e. Compile-time error
- f. Run-time error
- g. None of the above

### Question 5

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
    public static void main(String[] args) {
        A c1 = new C(); B c2 = new C(); C c3 = new C(); C c4 = new C();
        c4.m1(c1); c4.m1(c2); c4.m1(c3);
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: AAA
- b. Prints: ABC
- c. Prints: CCC
- d. Compile-time error
- e. Run-time error
- f. None of the above

### Question 6

```
class GFC217 {
    static String m(int i) {return "int";}
    static String m(float i) {return "float";}
    public static void main (String[] args) {
        long a1 = 1; double b1 = 2; System.out.print(m(a1)+", "+ m(b1));
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: float,float
- b. Prints: float,double
- c. Prints: double,float
- d. Prints: double,double
- e. Compile-time error
- f. Run-time error
- g. None of the above

### Question 7

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
    public static void main(String[] args) {
        A c1 = new C(); C c2 = new C(); c1.m1(c2);
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: A
- b. Prints: B
- c. Prints: C
- d. Compile-time error
- e. Run-time error
- f. None of the above

**Question 8**

```
class GFC218 {
    static void m(Object x) {System.out.print("Object");}
    static void m(String x) {System.out.print("String");}
    public static void main(String[] args) {m(null);}
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- |                   |                       |                      |
|-------------------|-----------------------|----------------------|
| a. Prints: Object | c. Compile-time error | e. None of the above |
| b. Prints: String | d. Run-time error     |                      |

**Question 9**

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
    public static void main(String[] args) {
        A a1 = new A(); A b1 = new B(); A c1 = new C(); C c4 = new C();
        a1.m1(c4); b1.m1(c4); c1.m1(c4);
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- |                |                       |                      |
|----------------|-----------------------|----------------------|
| a. Prints: AAA | c. Prints: CCC        | e. Run-time error    |
| b. Prints: ABC | d. Compile-time error | f. None of the above |

**Question 10**

```
class GFC200 {}
class GFC201 {
    static void m(Object x) {System.out.print("Object");}
    static void m(String x) {System.out.print("String");}
    static void m(GFC200 x) {System.out.print("GFC200");}
    public static void main(String[] args) {m(null);}
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- |                   |                       |                      |
|-------------------|-----------------------|----------------------|
| a. Prints: Object | c. Prints: GFC200     | e. Run-time error    |
| b. Prints: String | d. Compile-time error | f. None of the above |

**Question 11**

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
    public static void main(String[] args) {
        A a1 = new A(); B b1 = new B(); C c1 = new C(); A c2 = new C();
        c2.m1(a1); c2.m1(b1); c2.m1(c1);
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- |                |                       |                      |
|----------------|-----------------------|----------------------|
| a. Prints: AAA | c. Prints: CCC        | e. Run-time error    |
| b. Prints: ABC | d. Compile-time error | f. None of the above |

## 6 Chapter 12: Method Overloading, Exam 1

### Question 12

```
class GFC202 {} class GFC203 extends GFC202 {}
class GFC204 {
    static void m(GFC202 x) {System.out.print("GFC202");}
    static void m(GFC203 x) {System.out.print("GFC203");}
    public static void main(String[] args) {m(null);}
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: GFC202
- b. Prints: GFC203
- c. Compile-time error
- d. Run-time error
- e. None of the above

### Question 13

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
    public static void main(String[] args) {
        A a1 = new A(); B b1 = new A(); C c1 = new A(); C c2 = new C();
        c2.m1(a1); c2.m1(b1); c2.m1(c1);
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: AAA
- b. Prints: ABC
- c. Prints: CCC
- d. Compile-time error
- e. Run-time error
- f. None of the above

### Question 14

```
class GFC205 {} class GFC206 extends GFC205 {}
class GFC207 extends GFC206 {
    static void m(GFC205 x, GFC205 y) {System.out.print("GFC205,GFC205");}
    static void m(GFC205 x, GFC206 y) {System.out.print("GFC205,GFC206");}
    static void m(GFC206 x, GFC205 y) {System.out.print("GFC206,GFC205");}
    static void m(GFC206 x, GFC206 y) {System.out.print("GFC206,GFC206");}
    public static void main(String[] args) {
        GFC207 gfc207 = new GFC207(); m(gfc207, gfc207);
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: GFC205,GFC205
- b. Prints: GFC205,GFC206
- c. Prints: GFC206,GFC205
- d. Prints: GFC206,GFC206
- e. Compile-time error
- f. Run-time error
- g. None of the above

**Question 15**

```

class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
    public static void main(String[] args) {
        A a1 = new A(); B b1 = new B(); C c1 = new C(); C c2 = new A();
        c2.m1(a1); c2.m1(b1); c2.m1(c1);
    }
}

```

What is the result of attempting to compile and run the program? (Select 1.)

- |                |                       |                      |
|----------------|-----------------------|----------------------|
| a. Prints: AAA | c. Prints: CCC        | e. Run-time error    |
| b. Prints: ABC | d. Compile-time error | f. None of the above |

**Question 16**

```

class GFC211 {} class GFC212 extends GFC211 {}
class GFC213 extends GFC212 {
    static void m(GFC211 x, GFC211 y) {System.out.print("GFC211,GFC211");}
    static void m(GFC211 x, GFC212 y) {System.out.print("GFC211,GFC212");}
    static void m(GFC212 x, GFC211 y) {System.out.print("GFC212,GFC211");}
    static void m(GFC212 x, GFC212 y) {System.out.print("GFC212,GFC212");}
    static void m(GFC211 x, GFC213 y) {System.out.print("GFC211,GFC213");}
    public static void main(String[] args) {
        GFC213 gfc213 = new GFC213(); m(gfc213, gfc213);
    }
}

```

What is the result of attempting to compile and run the program? (Select 1.)

- |                          |                          |                      |
|--------------------------|--------------------------|----------------------|
| a. Prints: GFC211,GFC211 | d. Prints: GFC212,GFC212 | g. Run-time error    |
| b. Prints: GFC211,GFC212 | e. Prints: GFC211,GFC213 | h. None of the above |
| c. Prints: GFC212,GFC211 | f. Compile-time error    |                      |

**Question 17**

```

class GFC214 {
    static void m1(boolean b1) {System.out.print("boolean ");}
    static void m1(byte b1) {System.out.print("byte ");}
    static void m1(short s1) {System.out.print("short ");}
    static void m1(char c1) {System.out.print("char ");}
    static void m1(int i1) {System.out.print("int ");}
    public static void main(String[] args) {
        byte b1; m1(b1 = 1); m1(b1); m1(b1 == 1);
    }
}

```

What is the result of attempting to compile and run the program? (Select 1.)

- |                              |                        |                      |
|------------------------------|------------------------|----------------------|
| a. Prints: byte byte byte    | c. Prints: int int int | e. Run-time error    |
| b. Prints: byte byte boolean | d. Compile-time error  | f. None of the above |

## 8 Chapter 12: Method Overloading, Exam 1

### Question 18

```
class A {} class B extends A {}
class C extends B {
    static void m(A x, A y) {System.out.print("AA");}
    static void m(A x, B y) {System.out.print("AB");}
    static void m(B x, A y) {System.out.print("BA");}
    static void m(B x, B y) {System.out.print("BB");}
    public static void main(String[] args) {
        A a1; B b1; m(null,null); m(a1=null,b1=null); m(b1, a1);
    }
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: BBABAB
- b. Prints: BBABBA
- c. Prints: BBBBAB
- d. Prints: BBBBBA
- e. Prints: BBBBBB
- f. Compile-time error
- g. Run-time error
- h. None of the above

### Question 19

```
class A {} class B extends A {}
class C extends B {
    static void m1(A x) {System.out.print("m1A");}
    static void m2(B x) {System.out.print("m2B"); m1(x);}
    static void m2(A x) {System.out.print("m2A"); m1(x);}
    static void m3(C x) {System.out.print("m3C"); m2(x);}
    static void m3(B x) {System.out.print("m3B"); m2(x);}
    static void m3(A x) {System.out.print("m3A"); m2(x);}
    public static void main(String[] args) {m3(new C());}
}
```

What is the result of attempting to compile and run the program? (Select 1.)

- a. Prints: m3Am2Am1A
- b. Prints: m3Bm2Bm1A
- c. Prints: m3Cm2Bm1A
- d. Prints: m3Cm2Am1A
- e. Compile-time error
- f. Run-time error
- g. None of the above

# Answers: Exam 1

## Chapter 12: Method Overloading

---

### 1. b. Prints: ABC

The method invocation expression `d1.m1(a1)` uses reference `d1` of type `D` to invoke method `m1`. Since the reference `d1` is of type `D`, the class `D` is searched for an applicable implementation of `m1`. The methods inherited from the superclasses, `C`, `B` and `A`, are included in the search. The argument, `a1`, is a variable declared with the type `A`; so method `A.m1(A a)` is invoked.

### 2. a. Prints: float,float

A method invocation conversion can widen an argument of type `float` to match a method parameter of type `double`, so any argument that can be passed to `m(float i)` can also be passed to `m(double i)` without generating a compile-time type error. For that reason, we can say that `m(float i)` is more specific than `m(double i)`. Since both methods are applicable, the more specific of the two, `m(float i)`, is chosen over the less specific, `m(double i)`. The arguments of the method invocation expressions, `m(a1)` and `m(b1)`, are of types `int` and `long` respectively. A method invocation conversion can widen an argument of type `int` or `long` to match either of the two method parameter types `float` or `double`; so both methods, `m(float i)` and `m(double i)`, are applicable to the two method invocation expressions. Since both methods are applicable, the more specific of the two, `m(float i)` is chosen rather than the less specific, `m(double i)`.

### 3. b. Prints: ABC

Three methods overload the method name `m1`. Each has a single parameter of type `A` or `B` or `C`. For any method invocation expression of the form `m1(referenceArgument)`, the method is selected based on the declared type of the variable `referenceArgument`--not the run-time type of the referenced object. The method invocation expression `d1.m1(c1)` uses reference `d1` of type `D` to invoke method `m1` on an instance of type `D`. The argument, `c1`, is a reference of type `A` and the run-time type of the referenced object is `C`. The argument type is determined by the declared type of the reference variable `c1`--not the run-time type of the object referenced by `c1`. The declared type of `c1` is type `A`; so the method `m1(A a)` is selected. The declared type of `c2` is type `B`; so the method invocation expression `d1.m1(c2)` invokes method `m1(B b)`. The declared type of `c3` is type `C`; so the method invocation expression `d1.m1(c3)` invokes method `m1(C c)`.

### 4. a. Prints: float,float

A method invocation conversion can widen an argument of type `float` to match a method parameter of type `double`, so any argument that can be passed to `m(float i)` without generating a compile-time type error can also be passed to `m(double i)`. For that reason, we can say that `m(float i)` is more specific than `m(double i)`. The arguments of the method invocation expressions, `m(a1)` and `m(b1)`, are of types `char` and `long` respectively. A method invocation conversion can widen an argument of type `char` or `long` to match either of the two method parameter types `float` or `double`; so both methods, `m(float i)` and `m(double i)`, are applicable to the two method invocation expressions. Since both methods are applicable, the more specific of the two, `m(float i)` is chosen rather than the less specific, `m(double i)`.

### 5. b. Prints: ABC

Three methods overload the method name `m1`. Each has a single parameter of type `A` or `B` or `C`. For any method invocation expression of the form `m1(referenceArgument)`, the method is selected

## 10 Chapter 12: Method Overloading, Exam 1

based on the declared type of the variable `referenceArgument`--not the run-time type of the referenced object. The method invocation expression `c4.m1(c1)` uses reference `c4` of type `C` to invoke method `m1` on an instance of type `C`. The argument, `c1`, is a reference of type `A` and the run-time type of the referenced object is `C`. The argument type is determined by the declared type of the reference variable `c1`--not the run-time type of the object referenced by `c1`. The declared type of `c1` is type `A`; so the method `A.m1(A a)` is selected. The declared type of `c2` is type `B`; so the method invocation expression `c4.m1(c2)` invokes method `B.m1(B b)`. The declared type of `c3` is type `C`; so the method invocation expression `c4.m1(c3)` invokes method `C.m1(C c)`.

### 6. e. Compile-time error

The method invocation expression, `m(b1)`, contains an argument of type `double`. A method invocation conversion will not implicitly narrow the argument to match the parameter type of the method, `m(float i)`. The method invocation expression, `m(a1)`, contains an argument of type `long`. A method invocation conversion will widen the argument to match the parameter type of the the method, `m(float i)`.

### 7. a. Prints: A

The reference `c1` is of the superclass type, `A`; so it can be used to invoke only the method `m1` declared in class `A`. The methods that overload the method name `m1` in the subclasses, `B` and `C`, can not be invoked using the reference `c1`. A method invocation conversion promotes the argument referenced by `c2` from type `C` to type `A`, and the method declared in class `A` is executed. Class `A` declares only one method, `m1`. The single parameter is of type `A`. Class `B` inherits the method declared in class `A` and overloads the method name with a new method that has a single parameter of type `B`. Both methods sharing the overloaded name, `m1`, can be invoked using a reference of type `B`; however, a reference of type `A` can be used to invoke only the method declared in class `A`. Class `C` inherits the methods declared in classes `A` and `B` and overloads the method name with a new method that has a single parameter of type `C`. All three methods sharing the overloaded name, `m1`, can be invoked using a reference of type `C`; however, a reference of type `B` can be used to invoke only the method declared in class `B` and the method declared in the superclass `A`. The method invocation expression `c1.m1(c2)` uses reference `c1` of type `A` to invoke method `m1`. Since the reference `c1` is of type `A`, the search for an applicable implementation of `m1` is limited to class `A`. The subclasses, `B` and `C`, will not be searched; so the overloading methods declared in the subclasses can not be invoked using a reference of the superclass type.

### 8. b. Prints: String

A method invocation conversion can widen an argument of type `String` to match a method parameter of type `Object`, so any argument that can be passed to `m(String x)` without generating a compile-time type error can also be passed to `m(Object x)`. For that reason, we can say that `m(String x)` is more specific than `m(Object x)`. The argument of the method invocation expression, `m(null)`, is of type `null` and can be converted to either type `String` or `Object` by method invocation conversion, so both methods, `m(String x)` and `m(Object x)`, are applicable. The more specific of the two, `m(String x)`, is chosen over the less specific, `m(Object x)`.

### 9. a. Prints: AAA

The declared type of the reference variables, `a1`, `b1` and `c1`, is the superclass type, `A`; so the three reference variables can be used to invoke only the method `m1(A a)` that is declared in the superclass, `A`. The methods that overload the method name `m1` in the subclasses, `B` and `C`, can not be invoked using a reference variable of the superclass type, `A`. A method invocation conversion promotes the argument referenced by `c4` from type `C` to type `A`, and the method declared in class `A` is executed.

**10. d. Compile-time error**

The type of the argument is `null` and could be converted to any of the types `Object`, `String` or `GFC200`, by method invocation conversion. All three methods are applicable, but none of the three is more specific than both of the other two. The ambiguity results in a compile-time type error. If type `GFC200` were a subclass of type `String`; then any argument that could be pass to `m(GFC200 x)` could also be passed to `m(String x)` without causing a compile-time type error, and we could say that `m(GFC200 x)` is more specific than `m(String x)`. Since `GFC200` is not a subclass of type `String`, a method invocation conversion is not able to widen an argument of type `GFC200` to match a method parameter of type `String`, so `m(GFC200 x)` is not more specific than `m(String x)`.

**11. a. Prints: AAA**

The reference `c2` is of the superclass type, `A`; so it can be used to invoke only the method, `m1`, declared in class `A`. The methods that overload the method name `m1` in the subclasses, `B` and `C`, can not be invoked using the reference `c2`.

**12. b. Prints: GFC203**

The type of the argument is `null` and could be converted to either type `GFC202` or `GFC203` by method invocation conversion; so both methods are applicable. The more specific of the two, `m(GFC203 x)`, is chosen. Type `GFC203` is a subclass of type `GFC202`; so any argument that can be passed to `m(GFC203 x)` can also be passed to method `m(GFC202 x)` without causing a compile-time type error; therefore, we can say that method `m(GFC203 x)` is more specific than `m(GFC202 x)`.

**13. d. Compile-time error**

The declarations of `b1` and `c1` cause compile-time errors, because a reference of a subclass type can not refer to an instance of the superclass type.

**14. d. Prints: GFC206,GFC206**

Type `GFC207` is a subclass of types `GFC206` and `GFC205`, so any of the four methods are applicable to the method invocation expression, `m(gfc207, gfc207)`. The most specific of the four, `m(GFC206 x, GFC206 y)`, is chosen. Type `GFC206` is a subclass of type `GFC205`, and method `m(GFC206 x, GFC206 y)` is more specific than the other three, because any invocation of `m(GFC206 x, GFC206 y)` could also be handled by any of the other three without causing a compile-time type error.

**15. d. Compile-time error**

The declaration of `c2` causes a compile-time error, because a reference of a subclass type can not refer to an instance of the superclass class.

**16. f. Compile-time error**

The method invocation expression, `m(gfc213, gfc213)`, is ambiguous; because, no applicable method is more specific than all of the others. Method `m(GFC212 x, GFC212 y)` is more specific than `m(GFC212 x, GFC211 y)`, because any invocation of `m(GFC212 x, GFC212 y)` could also be handled by `m(GFC212 x, GFC211 y)` without causing a compile-time type error. However, some invocations of `m(GFC212 x, GFC212 y)` can not be handled by `m(GFC211 x, GFC213 y)`, so `m(GFC212 x, GFC212 y)` is not more specific than `m(GFC211 x, GFC213 y)`. Furthermore, not all invocations of `m(GFC211 x, GFC213 y)` could be handled by `m(GFC212 x, GFC212 y)`.

**17. b. Prints: byte byte boolean**

## 12 Chapter 12: Method Overloading, Exam 1

Variable `b1` was initialized by the first method invocation statement, so the second method invocation statement does not result in a compile-time error. The assignment expression, `b1 = 1`, initializes variable `b1` with the value 1, and the same value is passed as an argument to method `m1(byte b1)`. The method invocation expression, `m1(b1)`, invokes the same method, `m1(byte b1)`. The argument of the third method invocation expression, `m1(b1 == 1)`, is the result of the equality expression, `b1 == 1`. The type of the result and the argument is `boolean`, so the invoked method is `m1(boolean b1)`.

### 18. b. Prints: BBABBA

Type `B` is a subclass of type `A`, and method `m(B x, B y)` is more specific than the other three; because any invocation of it could be handled by any of the other three without causing a compile-time type error. All four methods are applicable to the first method invocation expression, `m(null, null)`. The most specific method, `m(B x, B y)`, is chosen; and both arguments are converted to type `B`. In the second method invocation expression, `m(a1=null, b1=null)`, simple assignment expressions initialize the local variables `a1` and `b1` with `null` references of types `A` and `B` respectively. The invoked method is `m(A x, B y)`. In the third method invocation expression, the positions of the arguments are reversed relative to the previous invocation: The type of the first argument is now `B` and the second is `A`. The invoked method is `m(B x, A y)`.

### 19. c. Prints: m3Cm2Bm1A

The method invocation expression, `m3(new C())`, invokes method `m3(C x)`, because the argument type matches the parameter type of the method declaration exactly. Method `m3` uses the parameter as the argument of the next invocation expression, `m2(x)`. Of the two overloaded versions of `m2`, the most specific is invoked, `m2(B x)`. Type `B` is a subclass of `A`, so any invocation of `m2(B x)` could be handled by `m2(A x)` without causing a compile-time type error. For that reason, `m2(B x)` is more specific than `m2(A x)`.