class Eggs {

    int doX(Long x, Long y) { return 1;}

    int doX(long... x) { return 2;}

    int doX(Integer x, Integer y) { return 3; }

    int doX(Number n, Number m) { return 4; }

    public static void main(String[ ] args) {

        new Eggs().go();

    }

    void go() {

        short s = 7;

        System.out.println(doX(s,s) + " ");

        System.out.println(doX(7,7));

    }

}

1. 1  1
2. 2  1
3. 3  1
4. 4  1
5. 2  3
6. 3  3
7. 4  3

Ans: (G) “s” is a short variable and when doX(s,s) is called there is no method which takes short as argument so by autoboxing and unboxing concept short is converted in Short wrapper class which comes under abstract class Number so doX(Number n,Number m) is invoked and returns 4,the second doX(7,7) invokes integer wrapper class arguments given in third method.So, the answer becomes 4 3.

class Mixer {

    Mixer() { }

    Mixer(Mixer m) { m1 = m; }

    Mixer m1;

    public static void main(String[ ] args) {

        Mixer m2 = new Mixer();

        Mixer m3 = new Mixer(m2);

        m3.go();

        Mixer m4 = m3.m1;

        m4.go();

        Mixer m5 = m2.m1;

        m5.go();

    }

    void go() {

        System.out.println("hi ");

    }

}

1. hi
2. hi hi
3. hi hi hi
4. Compilation fails
5. hi, followed by an exception
6. hi hi, followed by an exception

Ans: (F) this prints “hi” twice followed by exception. When m5.go() is given m5 contains null reference so it throws an exception.

class Alien {

    String invade(short ships) {

        return " a few";

    }

}

class Defender {

    public static void main(String[] args) {

        System.out.println(new Alien().invade(7));

    }

}

1. many
2. a few
3. Compilation fails
4. The output is not predictable
5. An exception is thrown at runtime

Ans:(C) Compilation fails because the invade method takes short datatype as argument and a int datatype is passed while invoking.

class Fizz {

    int x = 5;

    public static void main(String[] args) {

        final Fizz f1 = new Fizz();

        Fizz f2 = new Fizz();

        Fizz f3 = FizzSwitch(f1, f2);

        System.out.println((f1 == f3) + " " + (f1.x == f3.x));

    }

    static Fizz FizzSwitch(Fizz x, Fizz y) {

        final Fizz z = x;

        z.x = 6;

        return z;

    }

}

1. true true
2. false true
3. true false
4. false false
5. Compilation fails.
6. An exception is thrown at runtime.

Ans: (A) final Fizz z=x makes the reference value as same of x i.e., f1 therefore f1==f3 returns true.

f1.x==f3.x returns true a sint x=5.

class Knowing {

    static final long tooth = 343L;

    static long doIt(long tooth) {

        System.out.print(++tooth + " ");

        return ++tooth;

    }

    public static void main(String[] args) {

        System.out.print(tooth + " ");

        final long tooth = 340L;

        new Knowing().doIt(tooth);

        System.out.println(tooth);

    }

}

1. 343  340  340
2. 343  340  342
3. 343  341  342
4. 343  341  340
5. 343  341  343
6. Compilation fails.
7. An exception is thrown at runtime.

Ans: (C) the first sop prints tooth=343 because static final long tooth =343L declared. Then after knowing method takes 340L as input so prints 341. The 3rd sop prints 340 as final long tooth=340L is declared.

class Bird {

    { System.out.print("b1 "); }

    public Bird() {

        System.out.print("b2 ");

    }

}

class Raptor extends Bird {

    static { System.out.print("r1 "); }

    public Raptor() {

        System.out.print("r2 ");

    }

    { System.out.print("r3 "); }

    static { System.out.print("r4 "); }

}

class Hawk extends Raptor {

    public static void main(String[] args) {

        System.out.print("pre ");

        new Hawk();

        System.out.println("hawk ");

    }

}

1. pre b1 b2 r3 r2 hawk
2. pre b2 b1 r2 r3 hawk
3. pre b2 b1 r2 r3 hawk r1 r4
4. r1 r4 pre b1 b2 r3 r2 hawk
5. r1 r4 pre b2 b1 r2 r3 hawk
6. pre r1 r4 b1 b2 r3 r2 hawk
7. pre r1 r4 b2 b1 r2 r3 hawk
8. The order of output cannot be predicted.
9. Compilation fails.

Ans: (F) first pre is printed, because of new Hawk() the default constructor of Raptor is invoked because of that default constructor of bird is invoked. In inheritance it is always top to bottom approach so first the static blocks present in raptor are printed,then compound block followed by default constructor of bird is printed,then the raptor class compound block followed by default constructor, comes to the main method and prints “hawk” at the end.

1. “X extends Y” is correct if and only if X is a class and Y is an interface.
2. “X extends Y” is correct if and only if X is an interface and Y is a class.
3. “X extends Y” is correct if X and Y are either both classes or both interfaces.
4. “X extends Y” is correct for all combinations of X and Y being classes and/or interfaces.

Ans: (C)

class Top {

    public Top(String s) {

        System.out.println("B");

    }

    public class Bottom2 extends Top {

        public Bootom2(String s) {

            System.out.println("D");

        }

        public static void main(String[] args) {

            new Bottom2("C");

            System.out.println(" ");

        }

    }

}

1. BD
2. DB
3. BDC
4. DBC
5. Compilation fails.

Ans: (E) the default constructor of Top is not defined.

class Clidders {

    public final void flipper() {

        System.out.println("Clidder");

    }

}

public class Clidlets  extends Clidders {

    public void flipper() {

        System.out.println("Flip a Clidlet");

        super.flipper();

    }

    public static void main(String[] args) {

        new Clidlets().flipper();

    }

}

1. Flip a Clidlet
2. Flip  a Clidder
3. Flip a Clidder

Flip a Clidlet

1. Flip a Clidlet

Flip a Clidder

1. Compilation fails.

Ans: (E) because the flipper() in Clidders is declared final and it can’t be overridden.

1. class Programmer {

2.     Programmer debug() { return this; }

3. }

4. class SCJP extends Programmer {

5.     // insert code here

6.}

Which method at line 5, will compile? (Choose all that apply.)

1. Programmer debug() { return this; }
2. SCJP debug()  { return this; }
3. Object debug() { return this; }
4. int debug() { return 1; }
5. int debug(int x) { return 1; }
6. Object debug(int x) { return this; }

Ans: (A)

class Clidder {

    private final void flipper() {

        System.out.println("Clidder");

    }

}

public class Clidlet  extends Clidder {

    public final void flipper() {

        System.out.println("Clidlet");

    }

    public static void main(String[] args) {

        new Clidlet().flipper();

    }

}

1. Clidlet
2. Clidder
3. Clidder

Clidlet

1. Clidlet

Clidder

1. Compilation fails.

Ans: (A)

 i.     interface Base {

ii.         boolean m1 ();

iii.         byte m2(short s);

iv.     }

Which code fragments will compile? (Choose all that apply.)

a)  interface Base2 implements Base { }

b)  abstract class Class2 extends Base {

c)  public boolean m1 ()  { return true; }  }

d)  abstract class Class2 implements Base { }

e)  abstract class Class2 implements Base {

f)  public boolean m1 () { return (true); }  }

g)  class Class2 implements Base {

h)  boolean m1 () { return false; }

i)  byte m2 (short s) { return 42; } }

Ans: (g,h,i)

a)  public abstract class Canine { public Bark speak(); }

b)  public abstract class Canine { public Bark speak() { } }

c)  public class Canine { public abstract Bark speak(); }

d)  public class Canine abstract { public abstract Bark speak(); }

Ans: (b,d)

Given:

public abstract interface Frobnicate { public void twiddle(String s); }

Which is a correct class?

a)  public abstract void twiddle(String s) { }

}

b)  public abstract class Frob implements Frobnicate { }

c)  public class Frob extends Frobnicate {

d)              public void twiddle(Integer i) { }

}

e)  public class Frob implements Frobnicate {

f)             public void twiddle(Integer i) { }

}

g)  public class Frob implements Frobnicate {

h)             public void twiddle(String i) { }

i)             public void twiddle(Integer s) { }

Ans: (c,d)

1. HAS-A relationships always rely on inheritance.
2. HAS-A relationships always rely on instance variables.
3. HAS-A relationships always require at least two class types.
4. HAS-A relationships always rely on polymorphism.
5. HAS-A relationships are always tightly coupled.

Ans: (B)

Given:

1. class Plant {

2.     String getName() ) { return “plant”; }

3.     Plant getType() ) { return this; }

4. }

5. class Flower extends Plant {

6.     // insert code here

7. }

8. class Tulip extends Flower { }

Which statement()s), inserted at line 6, will compile? ()Choose all that apply.)

1. **Flower getType() ) { return this; }**
2. String getType() ) { return “this”; }
3. **Plant getType() ) { return this; }**
4. **Tulip getType() ) { return new Tulip() ); }**

Ans:(A)

Given:

1.      class Zing {

2.      protected Hmpf h;

3. }

4. class Woop extends Zing {  }

5. class Hmpf { }

Which is true?

A.     Woop IS-A Hmpf and HAS-A zing.

B.     Zing IS-A Woop and HAS-A Hmpf.

C.     Hmpf HAS-A Woop and Woop IS-A Zing.

D.    Woop HAS-A Hmpf and Woop IS-A Zing.

E.     Zing HAS-A Hmpf and Zing IS-A Woop.

Ans:(B)

What is the output of the following program?

public abstract class AbstractTest {

    public int getNum()) {

        return 45;

    }

    public abstract class Bar {

        public int getNum()) {

            return 38;

        }

    }

    public static void main()String[] args) {

        AbstractTest t = new AbstractTest()) {

            public int getNum()) {

                return 22;

            }

        };

        AbstractTest.Bar f = t.new Bar()) {

            public int getNum()) {

                return 57;

            }

        };

        System.out.println()f.getNum()) + " " + t.getNum()));

    }

}

1. 57  22
2. 45  38
3. 45  57
4. An exception occurs at runtime.
5. Compilation fails.

Ans:(E)