OCA JAVA SE 8 PROGRAMMER

PRESENTED BY MR MARK LOVATT

15/10/2018

Your fastest way to learn. Guaranteed.

FIREBRAND

Although StayAhead Training Limited makes every effort to ensure that the information in this manual is correct, it does not accept any liability for inaccuracy or omission.

StayAhead Training does not accept any liability for any damages, or consequential damages, resulting from any information provided within this manual.

OCA Java SE8 Programmer I 1Z0-808

OCA 8 Exam Objectives (1)

Oracle Stated Objective
Java Basics
Define the scope of variables
Define the structure of a Java class
Create executable Java applications with a main method; run a Java program from the command line; including console output.
Import other Java packages to make them accessible in your code
Compare and contrast the features and components of Java such as: platform independence, object orientation, encapsulation, etc.
Working with Java Data Types
Declare and initialize variables (including casting of primitive data types)
Differentiate between object reference variables and primitive variables
Know how to read or write to object fields
Explain an Object's Lifecycle (creation, "dereference by reassignment" and garbage collection)
Develop code that uses wrapper classes such as Boolean, Double, and Integer

OCA 8 Exam Objectives (2)

Oracle Stated Objective
Using Operators and Decision Constructs
Use Java operators; including parentheses to override operator precedence
Test equality between Strings and other objects using == and equals()
Create if and if/else and ternary constructs
Use a switch statement
Creating and Using Arrays
Declare, instantiate, initialize and use one- and multi-dimensional arrays
Working with Methods and Encapsulation
Create methods with arguments and return values; including overloaded methods
Apply the static keyword to methods and fields
Create and overload constructors; including impact on default constructors
Apply access modifiers
Apply encapsulation principles to a class
Determine the effect upon object references and primitive values when they are passed into methods that change the values

OCA 8 Exam Objectives (3)

Oracle Stated Objective
Working with Inheritance
Describe inheritance and its benefits
Develop code that demonstrates the use of polymorphism; including overriding and object type versus reference type
Determine when casting is necessary
Use super and this to access objects and constructors
Use abstract classes and interfaces
Handling Exceptions
Differentiate among checked exceptions, unchecked exceptions, and Errors
Create a try-catch block and determine how exceptions alter normal program flow
Describe the advantages of Exception handling
Create and invoke a method that throws an exception
Recognize common exception classes (such as NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException, ClassCastException)

OCA 8 Exam Objectives (4)

Oracle Stated Objective
Working with Selected classes from the Java API
Manipulate data using the StringBuilder class and its methods
Creating and manipulating Strings
Create and manipulate calendar data using classes from java.time.LocalDateTime, java.time.LocalDate, java.time.LocalTime, java.time.format.DateTimeFormatter, java.time.Period
Declare and use an ArrayList of a given type
Write a simple Lambda expression that consumes a Lambda Predicate expression

OCA 8 Exam Prep

Section 0:
Initial Assessment Test

Initial Assessment Test Q1

What is the result of the following class? (Choose all that apply)

```
1: public class _C {
2:  private static int $;
3:  public static void main(String[] main) {
4:    String a_b;
5:    System.out.print($);
6:    System.out.print(a_b);
7:  } }
```

A. Compiler error on line 1.

B. Compiler error on line 2.

C. Compiler error on line 4.

D. Compiler error on line 5.

E. Compiler error on line 6.

F. `0null`

G. `nullnull`

Initial Assessment Test Q2

What is the result of the following code?

```
String s1 = "Java";
String s2 = "Java";
StringBuilder sb1 = new StringBuilder();
sb1.append("Ja").append("va");
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));
System.out.println(sb1.toString() == s1);
System.out.println(sb1.toString().equals(s1));
```

A. `true` is printed out exactly once.

B. `true` is printed out exactly twice.

C. `true` is printed out exactly three times.

D. `true` is printed out exactly four times.

E. The code does not compile.

Initial Assessment Test Q3

What is the output of the following code? (Choose all that apply)

```
1: interface HasTail { int getTailLength(); }
2: abstract class Puma implements HasTail {
3:   protected int getTailLength() {return 4;}
4: }
5: public class Cougar extends Puma {
6:    public static void main(String[] args) {
7:      Puma puma = new Puma();
8:      System.out.println(puma.getTailLength());
9:    }
10:
11:  public int getTailLength(int length) {return 2;}
12: }
```

A. 2

B. 4

C. The code will not compile because of line 3.

D. The code will not compile because of line 5.

E. The code will not compile because of line 7.

F. The code will not compile because of line 11.

G. The output cannot be determined from the code provided.

Initial Assessment Test Q4

What is the output of the following program?

```
1: public class FeedingSchedule {
2:  public static void main(String[] args) {
3:    boolean keepGoing = true;
4:    int count = 0;
5:    int x = 3;
6:    while(count++ < 3) {
7:      int y = (1 + 2 * count) % 3;
8:      switch(y) {
9:         default:
10:         case 0: x -= 1; break;
11:         case 1: x += 5;
12:      }
13:    }
14:  System.out.println(x);
15: } }
```

A. 4

B. 5

C. 6

D. 7

E. 13

F. The code will not compile because of line 7.

Initial Assessment Test Q5

What is the output of the following code snippet?

```
13: System.out.print("a");
14: try {
15:   System.out.print("b");
16:   throw new IllegalArgumentException();
17: } catch (RuntimeException e) {
18:   System.out.print("c");
19: } finally {
20:   System.out.print("d");
21: }
22: System.out.print("e");
```

A. abe

B. abce

C. abde

D. abcde

E. The code does not compile.

F. An uncaught exception is thrown.

Initial Assessment Test Q6

What is the result of the following program?

```
1: public class MathFunctions {
2:   public static void addToInt(int x, int amountToAdd) {
3:     x = x + amountToAdd;
4:   }
5:   public static void main(String[] args) {
6:     int a = 15;
7:     int b = 10;
8:     MathFunctions.addToInt(a, b);
9:     System.out.println(a);   } }
```

A. `10`

B. `15`

C. `25`

D. Compiler error on line 3.

E. Compiler error on line 8.

F. None of the above.

Initial Assessment Test Q7

What is the result of the following code?

```
int[] array = {6,9,8};
List<Integer> list = new ArrayList<>();
list.add(array[0]);
list.add(array[2]);
list.set(1, array[1]);
list.remove(0);
System.out.println(list);
```

A. `[8]`

B. `[9]`

C. Something like `[Ljava.lang.String;@160bc7c0`

D. An exception is thrown.

E. The code does not compile.

Initial Assessment Test Q8

What is the output of the following code?

```
1: public class Deer {
2:  public Deer() { System.out.print("Deer"); }
3:  public Deer(int age) { System.out.print("DeerAge"); }
4:  private boolean hasHorns() { return false; }
5:  public static void main(String[] args) {
6:    Deer deer = new Reindeer(5);
7:    System.out.println(","+deer.hasHorns());
8:  }
9: }
10: class Reindeer extends Deer {
11:  public Reindeer(int age) { System.out.print("Reindeer"); }
12:  public boolean hasHorns() { return true; }
13: }
```

A. `DeerReindeer,false`
B. `DeerReindeer,true`
C. `ReindeerDeer,false`
D. `ReindeerDeer,true`
E. `DeerAgeReindeer,false`
F. `DeerAgeReindeer,true`
G. The code will not compile because of line 7.
H. The code will not compile because of line 12.

Initial Assessment Test Q9

Which of the following statements are true? (Choose all that apply)

A. Checked exceptions are intended to be thrown by the JVM (and not the programmer).
B. Checked exceptions are required to be caught or declared.
C. Errors are intended to be thrown by the JVM (and not the programmer).
D. Errors are required to be caught or declared.
E. Runtime exceptions are intended to be thrown by the JVM (and not the programmer).
F. Runtime exceptions are required to be caught or declared.

Initial Assessment Test Q10

Which are true of the following code? (Choose all that apply)

```
1:  import java.util.*;
2:  public class Grasshopper {
3:  public Grasshopper(String n) {
4:     name = n;
5:  }
6:  public static void main(String[] args) {
7:    Grasshopper one = new Grasshopper("g1");
8:    Grasshopper two = new Grasshopper("g2");
9:    one = two;
10:   two = null;
11:   one = null;
12: }
13:   private String name; }
```

A. Immediately after line 9, no grasshopper objects are eligible for garbage collection.

B. Immediately after line 10, no grasshopper objects are eligible for garbage collection.

C. Immediately after line 9, only one grasshopper object is eligible for garbage collection.

D. Immediately after line 10, only one grasshopper object is eligible for garbage collection.

E. Immediately after line 11, only one grasshopper object is eligible for garbage collection.

F. The code compiles.

G. The code does not compile.

Initial Assessment Test Q11

What is the output of the following program?

```
1: public class FeedingSchedule {
2:  public static void main(String[] args) {
3:    int x = 5, j = 0;
4:    OUTER: for(int i=0; i<3; )
5:      INNER: do {
6:        i++; x++;
7:        if(x > 10) break INNER;
8:        x += 4;
9:        j++;
10:     } while(j <= 2);
11:   System.out.println(x);
12: } }
```

A. 10

B. 12

C. 13

D. 17

E. The code will not compile because of line 4.

F. The code will not compile because of line 6.

Initial Assessment Test Q12

What is the result of the following program?

```
1: public class Egret {
2:   private String color;
3:   public Egret() {
4:     this("white");
5:   }
6:   public Egret(String color) {
7:     color = color;
8:   }
9:   public static void main(String[] args) {
10:    Egret e = new Egret();
11:    System.out.println("Color:" + e.color);
12:   }
13: }
```

A. `Color:`

B. `Color:null`

C. `Color:White`

D. Compiler error on line 4.

E. Compiler error on line 10.

F. Compiler error on line 11.

Initial Assessment Test Q13

What is the output of the following program?

```
1: public class BearOrShark {
2:   public static void main(String[] args) {
3:     int luck = 10;
4:     if((luck>10 ? luck++: --luck)<10) {
5:       System.out.print("Bear");
6:     } if(luck<10) System.out.print("Shark");
7: } }
```

A. `Bear`

B. `Shark`

C. `BearShark`

D. The code will not compile because of line 4.

E. The code will not compile because of line 6.

F. The code compiles without issue but does not produce any output.

Initial Assessment Test Q14

Assuming we have a valid, non-null HenHouse object whose value is initialized by the blank line shown here, which of the following are possible outputs of this application? (Choose all that apply)

```
1: class Chicken {}
2: interface HenHouse { public java.util.List<Chicken> getChickens(); }
3: public class ChickenSong {
4:   public static void main(String[] args) {
5:     HenHouse house = _______________
6:     Chicken chicken = house.getChickens().get(0);
7:     for(int i=0; i<house.getChickens().size();
8:       chicken = house.getChickens().get(i++)) {
9:       System.out.println("Cluck");
10: } } }
```

A. The code will not compile because of line 6.

B. The code will not compile because of lines 7–8.

C. The application will compile but not produce any output.

D. The application will output `Cluck` exactly once.

E. The application will output `Cluck` more than once.

F. The application will compile but produce an exception at runtime.

Initial Assessment Test Q15

Which of the following statements can be inserted in the blank line so that the code will compile successfully? (Choose all that apply)

```
public interface CanSwim {}
public class Amphibian implements CanSwim {}
class Tadpole extends Amphibian {}
public class FindAllTadPole {
  public static void main(String[] args) {
    List<Tadpole> tadpoles = new ArrayList<Tadpole>();
    for(Amphibian amphibian : tadpoles) {
      __________ tadpole = amphibian;
} } }
```

A. `CanSwim`

B. `Long`

C. `Amphibian`

D. `Tadpole`

E. `Object`

Initial Assessment Test Q16

What individual changes, if any, would allow the following code to compile? (Choose all that apply)

```
1: public interface Animal { public default String getName() { return null; } }
2: interface Mammal { public default String getName() { return null; } }
3: abstract class Otter implements Mammal, Animal {}
```

A. The code compiles without issue.

B. Remove the `default` method modifier and method implementation on line 1.

C. Remove the `default` method modifier and method implementation on line 2.

D. Remove the `default` method modifier and method implementation on lines 1 and 2.

E. Change the return value on line 1 from `null` to `"Animal"`.

F. Override the `getName()` method with an `abstract` method in the `Otter` class.

G. Override the `getName()` method with a concrete method in the `Otter` class.

Initial Assessment Test Q17

Which of the following lines can be inserted at line 11 to print `true`? (Choose all that apply)

```
10: public static void main(String[] args) {
11:   // INSERT CODE HERE
12: }
13: private static boolean test(Predicate<Integer> p) {
14:   return p.test(5);
15: }
```

A. `System.out.println(test(i -> i == 5));`

B. `System.out.println(test(i -> {i == 5;}));`

C. `System.out.println(test((i) -> i == 5));`

D. `System.out.println(test((int i) -> i == 5);`

E. `System.out.println(test((int i) -> {return i == 5;}));`

F. `System.out.println(test((i) -> {return i == 5;}));`

Initial Assessment Test Q18

Which of the following print out a date representing April 1, 2015? (Choose all that apply)

A. `System.out.println(LocalDate.of(2015, Calendar.APRIL, 1));`

B. `System.out.println(LocalDate.of(2015, Month.APRIL, 1));`

C. `System.out.println(LocalDate.of(2015, 3, 1));`

D. `System.out.println(LocalDate.of(2015, 4, 1));`

E. `System.out.println(new LocalDate(2015, 3, 1));`

F. `System.out.println(new LocalDate(2015, 4, 1));`

Initial Assessment Test Q19

Bytecode is in a file with which extension?

A. `.bytecode`

B. `.bytes`

C. `.class`

D. `.exe`

E. `.javac`

F. `.java`

Initial Assessment Test Q20

Which of the following are checked exceptions? (Choose all that apply)

A. `Exception`

B. `IllegalArgumentException`

C. `IOException`

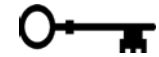
D. `NullPointerException`

E. `NumberFormatException`

F. `StackOverflowError`

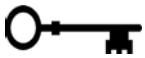
OCA 8 Exam Prep

Section 1:
Java Basics


Key Points

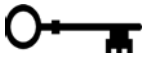
Identifiers

- Identifiers can begin with a letter, underscore or currency symbol
- After the first character, identifiers can also contain digits
- Indentifiers can be of any length



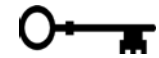
Executable Java Files and main()

- Compile with javac, execute with java command-line programs – both have many optional arguments
- The only versions of main() which are executed are public static void main(String[] args)
- main() can be overloaded



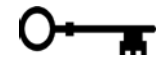
Imports

- import statements just save keystrokes
- Use an asterisk (*) to search through the contents of a single package
- Although referred to as static imports the syntax is import static
- You can import API classes and/or custom classes



Source File Declaration Rules

- A source file can only have one public class and multiple non-public classes
- If a source file contains a public class the filename must match the class name
- Files with no public classes have no naming restrictions
- A file can have only one package statement but multiple imports
- The order is: package (optional), import (optional), class
- package and import statements apply to all classes in the file



Stack and Heap

- Local variables (method variables) live on the stack.
- Objects and their instance variables live on the heap.

Literals and Primitive Casting

- Integer literals can be binary, decimal, octal (such as `013`), or hexadecimal (such as `0x3d`).
- Literals for `long`s end in `L` or `l`.
- Float literals end in `F` or `f`, and `double` literals end in a digit or `D` or `d`.
- The `boolean` literals are `true` and `false`.
- Literals for `char`s are a single character inside single quotes: `'d'`.

Scope

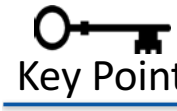
- Scope refers to the lifetime of a variable.
- There are four basic scopes:
  - Static variables live basically as long as their class lives.
  - Instance variables live as long as their object lives.
  - Local variables live as long as their method is on the stack; however, if their method invokes another method, they are temporarily unavailable.
  - Block variables (for example, in a `for` or an `if`) live until the block completes.

Basic Assignments

- Literal integers are implicitly `int`s.
- Integer expressions always result in an `int`-sized result, never smaller.
- Floating-point numbers are implicitly doubles (64 bits).
- Narrowing a primitive truncates the *high order* bits.
- Compound assignments (such as `+=`) perform an automatic cast.
- A reference variable holds the bits that are used to refer to an object.
- Reference variables can refer to subclasses of the declared type but not to superclasses.
- When you create a new object, such as `Button b = new Button();`, the JVM does three things:
  - Makes a reference variable named `b`, of type `Button`.
  - Creates a new `Button` object.
  - Assigns the `Button` object to the reference variable `b`.

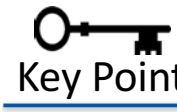
Unititialised/unassigned variables

- When an array of objects is instantiated, objects within the array are not instantiated automatically, but all the references get the default value of `null`.
- When an array of primitives is instantiated, elements get default values.
- Instance variables are always initialized with a default value.
- Local/automatic/method variables are never given a default value. If you attempt to use one before initializing it, you'll get a compiler error.



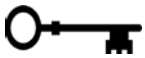
Passing variables into methods

- Methods can take primitives and/or object references as arguments.
- Method arguments are always copies.
- Method arguments are never actual objects (they can be references to objects).
- A primitive argument is an unattached copy of the original primitive.
- A reference argument is another copy of a reference to the original object.
- Shadowing occurs when two variables with different scopes share the same name. This leads to hard-to-find bugs and hard-to-answer exam questions.



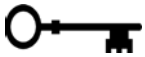
Garbage Collection

- In Java, garbage collection (GC) provides automated memory management.
- The purpose of GC is to delete objects that can't be reached.
- Only the JVM decides when to run the GC; you can only suggest it.
- You can't know the GC algorithm for sure.
- Objects must be considered eligible before they can be garbage collected.
- An object is eligible when no live thread can reach it.
- To reach an object, you must have a live, reachable reference to that object.
- Java applications can run out of memory.
- Islands of objects can be garbage collected, even though they refer to each other.
- Request garbage collection with `System.gc();` (for OCP 5 candidates only).
- The `Class` object has a `finalize()` method.
- The `finalize()` method is guaranteed to run once and only once before the garbage collector deletes an object.
- The garbage collector makes no guarantees; `finalize()` may never run.
- You can ineligible-ize an object for GC from within `finalize()`.



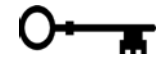
Local Variables

- Local (method, automatic or stack) variable declarations cannot have modifiers except final
- Local variables don't get auto-initialised and need to be initialised in code before use



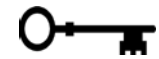
Methods with var-args

- As of Java 5, methods can declare a parameter that accepts from zero to many arguments, a so-called var-arg method.
- A var-arg parameter is declared with the syntax type... name; for instance: doStuff(int... x) { }.
- A var-arg method can have only one var-arg parameter.
- In methods with normal parameters and a var-arg, the var-arg must come last



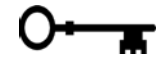
Variable Declarations (1)

- Instance variables can
  - Have any access control
  - Be marked final or transient
- Instance variables can't be abstract, synchronized, native, or strictfp.
- It is legal to declare a local variable with the same name as an instance variable; this is called "shadowing."
- The transient modifier applies only to instance variables.
- The volatile modifier applies only to instance variables



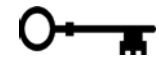
Variable Declarations (2)

- final variables:
  - cannot be reassigned once assigned a value
  - final reference variables cannot refer to a different object once the object has been assigned to the final variable
  - must be initialized before the constructor completes
- There is no such thing as a final object. An object reference marked final does NOT mean the object itself can't change



Static Variables and Methods

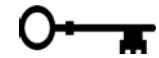
- They are not tied to any particular instance of a class
- No class instances are needed in order to use static members of the class
- There is only one copy of a static variable/class and all instances share it
- static methods do not have direct access to nonstatic members



Initialization Order

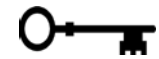
```
public class Egg {
   private int number = 3;
   public Egg() {
      number = 5;
   }
   public static void main(String[] args) {
      Egg egg = new Egg();
      System.out.println(egg.number);
   }
   { number = 4; }
}
```

- Fields and blocks are run first in order, setting number to 3 and then 4
- Then the constructor runs, setting number to 5



Benefits of Java (1)

- Java has some key benefits needed for the exam:
- Object Oriented
  - all code is defined in classes
  - classes can be instantiated into objects
  - Java allows for functional programming within a class
  - object oriented is still the main organization of code
- Encapsulation
  - access modifiers protect data from unintended access
  - encapsulation key aspect of object-orientation
- Platform Independent
  - interpreted language
  - compiled to bytecode
  - compiled once rather than for each operating system



Benefits of Java (2)

- Robust
  - prevents memory leaks
  - manages memory on its own
  - does garbage collection automatically
  - bad memory management in C++ is a big source of errors in programs.
- Simple
  - simpler than C++
  - no pointers
  - no operator overloading
- Secure
  - code runs inside the JVM
  - creates a sandbox
  - hard for Java code to directly target the machine it is running on

Self Test 

Self Test (1)

Which of the following are true? (Choose all that apply)

A. `javac` compiles a `.class` file into a `.java` file.

B. `javac` compiles a `.java` file into a `.bytecode` file.

C. `javac` compiles a `.java` file into a `.class` file.

D. Java takes the name of the class as a parameter.

E. Java takes the name of the `.bytecode` file as a parameter.

F. Java takes the name of the `.class` file as a parameter.

Self Test 

Self Test (2)

Which of the following are true statements? (Choose all that apply)

A. Java allows operator overloading.

B. Java code compiled on Windows can run on Linux.

C. Java has pointers to specific locations in memory.

D. Java is a procedural language.

E. Java is an object-oriented language.

F. Java is a functional programming language.

Self Test 

Self Test (3)

What is the output of the following program?

```
1: public class WaterBottle {
2: private String brand;
3: private boolean empty;
4: public static void main(String[] args) {
5:   WaterBottle wb = new WaterBottle();
6:   System.out.print("Empty = " + wb.empty);
7:   System.out.print(", Brand = " + wb.brand);
8:  } }
```

A. Line 6 generates a compiler error.

B. Line 7 generates a compiler error.

C. There is no output.

D. `Empty = false, Brand = null`

E. `Empty = false, Brand =`

F. `Empty = null, Brand = null`

Self Test 

Self Test (4)

Which of the following are valid Java identifiers? (Choose all that apply)

A. `A$B`

B. `_helloWorld`

C. `true`

D. `java.lang`

E. `Public`

F. `1980_s`

Self Test 

Self Test (5)

What does the following code output?

```
1: public class Salmon {
2:  int count;
3:  public void Salmon() {
4:    count = 4;
5:  }
6: public static void main(String[] args) {
7:  Salmon s = new Salmon();
8:  System.out.println(s.count);
9: } }
```

A. 0

B. 4

C. Compilation fails on line 3.

D. Compilation fails on line 4.

E. Compilation fails on line 7.

F. Compilation fails on line 8.

Self Test 

Self Test (6)

What is true about the following code? (Choose all that apply)

```
public class Bear {
  protected void finalize() {
    System.out.println("Roar!");
}
public static void main(String[] args) {
  Bear bear = new Bear();
  bear = null;
  System.gc();
} }
```

A. `finalize()` is guaranteed to be called.

B. `finalize()` might or might not be called

C. `finalize()` is guaranteed not to be called.

D. Garbage collection is guaranteed to run.

E. Garbage collection might or might not run.

F. Garbage collection is guaranteed not to run.

G. The code does not compile.

Self Test 

Self Test (7)

Suppose we have a class named `Rabbit`. Which of the following statements are true? (Choose all that apply)

```
1:  public class Rabbit {
2:   public static void main(String[] args) {
3:    Rabbit one = new Rabbit();
4:    Rabbit two = new Rabbit();
5:    Rabbit three = one;
6:    one = null;
7:    Rabbit four = one;
8:    three = null;
9:    two = null;
10:   two = new Rabbit();
11:   System.gc();
12: } }
```

A. The `Rabbit` object from line 3 is first eligible for garbage collection immediately following line 6.

B. The `Rabbit` object from line 3 is first eligible for garbage collection immediately following line 8.

C. The `Rabbit` object from line 3 is first eligible for garbage collection immediately following line 12.

D. The `Rabbit` object from line 4 is first eligible for garbage collection immediately following line 9.

E. The `Rabbit` object from line 4 is first eligible for garbage collection immediately following line 11.

F. The `Rabbit` object from line 4 is first eligible for garbage collection immediately following line 12.

Self Test 

Self Test (8)

Which represent the order in which the following statements can be assembled into a program that will compile successfully? (Choose all that apply)

```
A: class Rabbit {}
B: import java.util.*;
C: package animals;
```

A. A, B, C

B. B, C, A

C. C, B, A

D. B, A

E. C, A

F. A, C

G. A, B

Self Test 

Self Test (9)

Which of the following are true? (Choose all that apply)

```
public class Bunny {
       public static void main(String[] args) {
            Bunny bun = new Bunny();
} }
```

A. Bunny is a class.

B. bun is a class.

C. main is a class.

D. Bunny is a reference to an object.

E. bun is a reference to an object.

F. main is a reference to an object.

G. None of the above.

Self Test 

Self Test (10)

Given:

```
 4. class Announce {
 5.   public static void main(String[] args) {
 6.     for(int __x = 0; __x < 3; __x++) ;
 7.     int #lb = 7;
 8.     long [] x [5];
 9.     Boolean []ba[];
10.   }
11. }
```

What is the result? (Choose all that apply.)

A. Compilation succeeds

B. Compilation fails with an error on line 6

C. Compilation fails with an error on line 7

D. Compilation fails with an error on line 8

E. Compilation fails with an error on line 9

Self Test 

Self Test (11)

Given the following class, which of the following lines of code can replace `INSERT CODE HERE` to make the code compile? (Choose all that apply)

```
public class Price {
      public void admission() {
                INSERT CODE HERE
              System.out.println(amount);
      } }
```

A. `int amount = 9L;`

B. `int amount = 0b101;`

C. `int amount = 0xE;`

D. `double amount = 0xE;`

E. `double amount = 1_2_.0_0;`

F. `int amount = 1_2_;`

G. None of the above.

Self Test 

Self Test (12)

Given that the Integer class is in the java.lang package, and given:

```
1. // insert code here
2. class StatTest {
3.   public static void main(String[] args) {
4.     System.out.println(Integer.MAX_VALUE);
5.   }
6. }
```

Which, inserted independently at line 1, compiles? (Choose all that apply.)

A. `import static java.lang;`

B. `import static java.lang.Integer;`

C. `import static java.lang.Integer.*;`

D. `static import java.lang.Integer.*;`

E. `import static java.lang.Integer.MAX_VALUE;`

F. None of the above statements are valid import syntax.

Self Test 

Self Test (13)

Which of the following lines of code compile? (Choose all that apply)

A. `int i1 = 1_234;`

B. `double d1 = 1_234_.0;`

C. `double d2 = 1_234._0;`

D. `double d3 = 1_234.0_;`

E. `double d4 = 1_234.0;`

F. None of the above.

Self Test 

Self Test (14)

Given the following class in the file `/my/directory/named/A/Bird.java`:

```
INSERT CODE HERE
public class Bird { }
```

Which of the following replaces `INSERT CODE HERE` if we compile from `/my/directory`? (Choose all that apply)

A. `package my.directory.named.a;`

B. `package my.directory.named.A;`

C. `package named.a;`

D. `package named.A;`

E. `package a;`

F. `package A;`

G. Does not compile.

Self Test 

Self Test (15)

Which of the following are true? (Choose all that apply)

A. An instance variable of type `boolean` defaults to `false`.

B. An instance variable of type `boolean` defaults to `true`.

C. An instance variable of type `boolean` defaults to `null`.

D. An instance variable of type `int` defaults to `0`.

E. An instance variable of type `int` defaults to `0.0`.

F. An instance variable of type `int` defaults to `null`.

G. None of the above.

Self Test 

Self Test (16)

Given that the `for` loop's syntax is correct, and given:

```
import static java.lang.System.*;
class _ {
static public void main(String[] __A_V_) {
  String $ = "";
  for(int x=0; ++x < __A_V_.length; )   // for loop
    $ += __A_V_[x];
  out.println($);
 }
}
```

And the command line:

```
java _ - A .
```

What is the result?

A. `-A`
B. `A.`
C. `-A.`
D. `_A.`
E. `_-A.`
F. Compilation fails
G. An exception is thrown at runtime

Self Test 

Self Test (17)

Which of the following are true? (Choose all that apply)

A. A local variable of type `boolean` defaults to `null`.

B. A local variable of type `float` defaults to `0`.

C. A local variable of type `Object` defaults to `null`.

D. A local variable of type `boolean` defaults to `false`.

E. A local variable of type `boolean` defaults to `true`.

F. A local variable of type `float` defaults to `0.0`.

G. None of the above.

Self Test 

Self Test (18)

Given two files:

```
1. package pkgA;
2. public class Foo {
3. int a = 5;
4. protected int b = 6;
5. public int c = 7;
6. }
```

```
3. package pkgB;
4. import pkgA.*;
5. public class Baz {
6.   public static void main(String[] args) {
7.     Foo f = new Foo();
8.     System.out.print(" " + f.a);
9.     System.out.print(" " + f.b);
10.    System.out.println(" " + f.c);
11.  }
12. }
```

What is the result? (Choose all that apply.)

A. `5 6 7`

B. `5` followed by an exception

C. Compilation fails with an error on line 7

D. Compilation fails with an error on line 8

E. Compilation fails with an error on line 9

F. Compilation fails with an error on line 10

Self Test 

Self Test (19)

Which of the following are true? (Choose all that apply)

A. An instance variable of type `double` defaults to `null`.

B. An instance variable of type `int` defaults to `null`.

C. An instance variable of type `String` defaults to `null`.

D. An instance variable of type `double` defaults to `0.0`.

E. An instance variable of type `int` defaults to `0.0`.

F. An instance variable of type `String` defaults to `0.0`.

G. None of the above.

Self Test 

Self Test (20)

Which of the following are legal entry point methods that can be run from the command line? (Choose all that apply)

A. `private static void main(String[] args)`

B. `public static final main(String[] args)`

C. `public void main(String[] args)`

D. `public static void test(String[] args)`

E. `public static void main(String[] args)`

F. `public static main(String[] args)`

G. None of the above.

Self Test 

Self Test (21)

Which of the following legally fill in the blank so you can run the `main()` method from the command line? (Choose all that apply)

```
public static void main(_______)
```

A. `String[] _names`

B. `String[] 123`

C. `String abc[]`

D. `String _Names[]`

E. `String... $n`

F. `String names`

G. None of the above.

Self Test 

Self Test (22)

Given the following class, which of the following calls print out `Blue Jay`? (Choose all that apply)

```
public class BirdDisplay {
  public static void main(String[] name) {
    System.out.println(name[1]);
} }
```

A. `java BirdDisplay Sparrow Blue Jay`

B. `java BirdDisplay Sparrow "Blue Jay"`

C. `java BirdDisplay Blue Jay Sparrow`

D. `java BirdDisplay "Blue Jay" Sparrow`

E. `java BirdDisplay.class Sparrow "Blue Jay"`

F. `java BirdDisplay.class "Blue Jay" Sparrow`

G. Does not compile.

Self Test 

Self Test (23)

Given the following classes, which of the following snippets can be inserted in place of `INSERT IMPORTS HERE` and have the code compile? (Choose all that apply)

```
package aquarium;
public class Water {
  boolean salty = false;
}
package aquarium.jellies;
public class Water {
  boolean salty = true;
}
package employee;
 INSERT IMPORTS HERE
public class WaterFiller {
  Water water;
}
```

A. `import aquarium.*;`

B.
```
import aquarium.Water;
import aquarium.jellies.*;
```

C.
```
import aquarium.*;
import aquarium.jellies.Water;
```

D.
```
import aquarium.*;
import aquarium.jellies.*;
```

E.
```
import aquarium.Water;
import aquarium.jellies.Water;
```

F. None of these imports can make the code compile.

Self Test 

Self Test (24)

Given the following classes, what is the maximum number of imports that can be removed and have the code still compile?

```
package aquarium; public class Water { }

package aquarium;
import java.lang.*;
import java.lang.System;
import aquarium.Water;
import aquarium.*;
public class Tank {
  public void print(Water water) {
   System.out.println(water); } }
```

A. 0

B. 1

C. 2

D. 3

E. 4

F. Does not compile.

Self Test 

Self Test (25)

Given the following classes, which of the following can independently replace `INSERT IMPORTS HERE` to make the code compile? (Choose all that apply)

```
package aquarium;
public class Tank { }

package aquarium.jellies;
public class Jelly { }

package visitor;
INSERT IMPORTS HERE
public class AquariumVisitor {
  public void admire(Jelly jelly) { } }
```

A. `import aquarium.*;`
B. `import aquarium.*.Jelly;`
C. `import aquarium.jellies.Jelly;`
D. `import aquarium.jellies.*;`
E. `import aquarium.jellies.Jelly.*;`
F. None of these can make the code compile.

Self Test 

Self Test (26)

Given the following class, which of the following is true? (Choose all that apply)

```
 1: public class Snake {
 2:
 3:  public void shed(boolean time) {
 4:
 5:    if (time) {
 6:
 7:    }
 8:    System.out.println(result);
 9:
10:  }
11: }
```

A. If `String result = "done";` is inserted on line 2, the code will compile.
B. If `String result = "done";` is inserted on line 4, the code will compile.
C. If `String result = "done";` is inserted on line 6, the code will compile.
D. If `String result = "done";` is inserted on line 9, the code will compile.
E. None of the above changes will make the code compile.

Self Test 

Self Test (27)

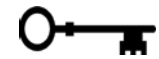
Which of the following are true? (Choose all that apply)

```
4: short numPets = 5;
5: int numGrains = 5.6;
6: String name = "Scruffy";
7: numPets.length();
8: numGrains.length();
9: name.length();
```

A. Line 4 generates a compiler error.
B. Line 5 generates a compiler error.
C. Line 6 generates a compiler error.
D. Line 7 generates a compiler error.
E. Line 8 generates a compiler error.
F. Line 9 generates a compiler error.
G. The code compiles as is.

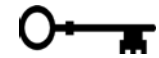
OCA 8 Exam Prep

Section 2:
Operators and Statements



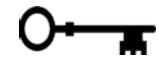
Relational Operators

- Relational operators always result in a `boolean` value (`true` or `false`).
- There are six relational operators: >, >=, <, <=, ==, and `!=`. The last two (== and `!=`) are sometimes referred to as *equality operators*.
- When comparing characters, Java uses the Unicode value of the character as the numerical value.
- Equality operators
  - There are two equality operators: == and `!=`.
  - Four types of things can be tested: numbers, characters, booleans, and reference variables.
- When comparing reference variables, == returns `true` only if both references refer to the same object.



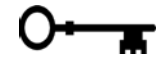
instanceof Operator

- `instanceof` is for reference variables only; it checks whether the object is of a particular type.
- The `instanceof` operator can be used only to test objects (or `null`) against class types that are in the same class hierarchy.
- For interfaces, an object passes the `instanceof` test if any of its superclasses implement the interface on the right side of the `instanceof` operator.



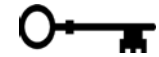
Arithmetic Operators

- The four primary math operators are add (`+`), subtract (`–`), multiply (`*`), and divide (`/`).
- The remainder (a.k.a. modulus) operator (`%`) returns the remainder of a division.
- Expressions are evaluated from left to right, unless you add parentheses, or unless some operators in the expression have higher precedence than others.
- The `*`, `/`, and `%` operators have higher precedence than `+` and `–`.



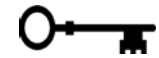
String Concatenation

- If either operand is a String, the + operator concatenates the operands
- If both operands are numeric, the + operator adds the operands.
- Expressions evaluated left to right
- Use round brackers to override precedence



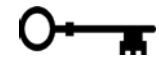
Increment/Decrement Operators

- Prefix operators (for example, `++x` and `--x`) run before the value is used in the expression.
- Postfix operators (for example, `x++` and `x--`) run after the value is used in the expression.
- In any expression, both operands are fully evaluated *before* the operator is applied.
- Variables marked `final` cannot be incremented or decremented.



Ternary Operator

- Returns one of two values based on whether its `boolean` expression is `true` or `false`.
  - Returns the value after the `?` if the expression is `true`.
  - Returns the value after the `:` if the expression is `false`.



Logical Operators

- The exam covers six "logical" operators: `&`, `|`, `^`, `!`, `&&`, and `||`.
- Logical operators work with two expressions (except for `!`) that must resolve to boolean values.
- The `&&` and `&` operators return `true` only if both operands are `true`.
- The `||` and `|` operators return `true` if either or both operands are `true`.
- The `&&` and `||` operators are known as short-circuit operators.
- The `&&` operator does not evaluate the right operand if the left operand is `false`.
- The `||` does not evaluate the right operand if the left operand is `true`.
- The `&` and `|` operators always evaluate both operands.
- The `^` operator (called the "logical XOR") returns `true` if exactly one operand is `true`.
- The `!` operator (called the "inversion" operator) returns the opposite value of the boolean operand it precedes.

if and switch statements (1)

- The only legal expression in an `if` statement is a `boolean` expression—in other words, an expression that resolves to a `boolean` or a `Boolean` reference.
- Watch out for `boolean` assignments (`=`) that can be mistaken for `boolean` equality (`==`) tests:

```
boolean x = false;
if (x = true) { } // an assignment, so x will always be true!
```

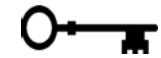
- Curly braces are optional for `if` blocks that have only one conditional statement. But watch out for misleading indentations.
- `switch` statements can evaluate only to `enum`s or the `byte`, `short`, `int`, `char`, and, as of Java 7, `String` data types. You can't say this:

```
long s = 30;
switch(s) { }
```

- The `case` constant must be a literal or `final` variable, or a constant expression, including an `enum` or a `String`. You cannot have a case that includes a non-final variable or a range of values.

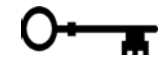
if and switch statements (2)

- If the condition in a `switch` statement matches a `case` constant, execution will run through all code in the `switch` following the matching `case` statement until a `break` statement or the end of the `switch` statement is encountered. In other words, the matching `case` is just the entry point into the `case` block, but unless there's a `break` statement, the matching `case` is not the only `case` code that runs.
- The `default` keyword should be used in a `switch` statement if you want to run some code when none of the `case` values match the conditional value.
- The `default` block can be located anywhere in the `switch` block, so if no preceding `case` matches, the `default` block will be entered, and if the `default` does not contain a `break`, then code will continue to execute (fall-through) to the end of the `switch` or until the `break` statement is encountered.



Loops (1)

- A basic `for` statement has three parts: declaration and/or initialization, boolean evaluation, and the iteration expression.
- If a variable is incremented or evaluated within a basic `for` loop, it must be declared before the loop or within the `for` loop declaration.
- A variable declared (not just initialized) within the basic `for` loop declaration cannot be accessed outside the `for` loop—in other words, code below the `for` loop won't be able to use the variable.
- You can initialize more than one variable of the same type in the first part of the basic `for` loop declaration; each initialization must be separated by a comma.
- An enhanced `for` statement (new as of Java 5) has two parts: the *declaration* and the *expression*. It is used only to loop through arrays or collections.
- With an enhanced `for`, the *expression* is the array or collection through which you want to loop.



Loops (2)

- With an enhanced `for`, the *declaration* is the block variable, whose type is compatible with the elements of the array or collection, and that variable contains the value of the element for the given iteration.
- You cannot use a number (old C-style language construct) or anything that does not evaluate to a `boolean` value as a condition for an `if` statement or looping construct. You can't, for example, say `if(x)`, unless `x` is a `boolean` variable.
- The `do` loop will enter the body of the loop at least once, even if the test condition is not met.

Key Points

break and continue

- An unlabeled `break` statement will cause the current iteration of the innermost looping construct to stop and the line of code following the loop to run.
- An unlabeled `continue` statement will cause the current iteration of the innermost loop to stop, the condition of that loop to be checked, and if the condition is met, the loop to run again.
- If the `break` statement or the `continue` statement is labeled, it will cause similar action to occur on the labeled loop, not the innermost loop.

Self Test

Self Test (1)

Given:

```
class Hexy {
  public static void main(String[] args) {
    int i = 42;
    String s = (i<40)?"life":(i>50)?"universe":"everything";
    System.out.println(s);
  }
}
```

What is the result?

A. `null`
B. `life`
C. `universe`
D. `everything`
E. Compilation fails
F. An exception is thrown at runtime

Self Test 

Self Test (2)

Given:

```
public class Dog {
  String name;
  Dog(String s) { name = s; }
  public static void main(String[] args) {
    Dog d1 = new Dog("Boi");
    Dog d2 = new Dog("Tyri");
    System.out.print((d1 == d2) + " ");
    Dog d3 = new Dog("Boi");
    d2 = d1;
    System.out.print((d1 == d2) + " ");
    System.out.print((d1 == d3) + " ");
  }
}
```

What is the result?

A. `true true true`
B. `true true false`
C. `false true false`
D. `false true true`
E. `false false false`
F. An exception will be thrown at runtime

Self Test 

Self Test (3)

Given:

```
class Fork {
  public static void main(String[] args) {
    if(args.length == 1 | args[1].equals("test")) {
      System.out.println("test case");
    } else {
      System.out.println("production " + args[0]);
    }
  }
}
```

And the command-line invocation:

```
java Fork live2
```

What is the result?

A. `test case`
B. `production live2`
C. `test case live2`
D. Compilation fails
E. An exception is thrown at runtime

Self Test 

Self Test (4)

Given:

```
class Feline {
  public static void main(String[] args) {
    long x = 42L;
    long y = 44L;
    System.out.print(" " + 7 + 2 + " ");
    System.out.print(foo() + x + 5 + " ");
    System.out.println(x + y + foo());
  }
  static String foo() { return "foo"; }
}
```

What is the result?

A. `9 foo47 86foo`
B. `9 foo47 4244foo`
C. `9 foo425 86foo`
D. `9 foo425 4244foo`
E. `72 foo47 86foo`
F. `72 foo47 4244foo`
G. `72 foo425 86foo`
H. `72 foo425 4244foo`
I. Compilation fails

Self Test 

Self Test (5)

Place the fragments into the code to produce the output `33`. Note that you must use each fragment exactly once.

```
CODE:
class Incr {
  public static void main(String[] args) {
    Integer x = 7;
    int y = 2;

    x    ___  ___;
    ___  ___  ___;
    ___  ___  ___;
    ___  ___  ___;

    System.out.println(x);
  }
}
```

FRAGMENTS:

```
y  y  y  y

y  x  x

-= *= *= *=
```

Self Test 

Self Test (6)

Given:

```
public class Cowboys {
  public static void main(String[] args) {
    int x = 12;
    int a = 5;
    int b = 7;
    System.out.println(x/a + " " + x/b);
  }
}
```

What is the result? (Choose all that apply.)

A. `2 1`
B. `2 2`
C. `3 1`
D. `3 2`
E. An exception is thrown at runtime

Self Test 

Self Test (8)

Given:

```
 4. public class SpecialOps {
 5.   public static void main(String[] args) {
 6.     String s = "";
 7.     boolean b1 = true;
 8.     boolean b2 = false;
 9.     if((b2 = false) | (21%5) > 2) s += "x";
10.     if(b1 || (b2 == true))        s += "y";
11.     if(b2 == true)                s += "z";
12.     System.out.println(s);
13.   }
14. }
```

Which are true? (Choose all that apply.)

A. Compilation fails
B. x will be included in the output
C. y will be included in the output
D. z will be included in the output
E. An exception is thrown at runtime

Self Test 

Self Test (9)

Given:

```
 3. public class Spock {
 4.   public static void main(String[] args) {
 5.     int mask = 0;
 6.     int count = 0;
 7.     if( ((5<7) || (++count < 10)) | mask++ < 10 )   mask = mask + 1;
 8.     if(  (6 > 8) ^ false)                           mask = mask + 10;
 9.     if( !(mask > 1) && ++count > 1)                 mask = mask + 100;
10.     System.out.println(mask + " " + count);
11.   }
12. }
```

Which two are true about the value of mask and the value of count at line 10? (Choose two.)

A. `mask` is 0
B. `mask` is 1
C. `mask` is 2
D. `mask` is 10
E. `mask` is greater than 10
F. `count` is 0
G. `count` is greater than 0

Self Test 

Self Test (10)

What is the result of the following code snippet?

```
3: final char a = 'A', d = 'D';
4: char grade = 'B';
5: switch(grade) {
6:   case a:
7:   case 'B': System.out.print("great");
8:   case 'C': System.out.print("good"); break;
9:   case d:
10:  case 'F': System.out.print("not good");
11: }
```

A. `great`
B. `greatgood`
C. The code will not compile because of line 3.
D. The code will not compile because of line 6.
E. The code will not compile because of lines 6 and 9.

Self Test 

Self Test (11)

What is the result of the following code snippet?

```
3: int m = 9, n = 1, x = 0;
4: while(m > n) {
5:    m--;
6:    n += 2;
7:    x += m + n;
8: }
9: System.out.println(x);
```

A. 11
B. 13
C. 23
D. 36
E. 50
F. The code will not compile because of line 7.

Self Test 

Self Test (12)

What is the output of the following code snippet?

```
3: int count = 0;
4: ROW_LOOP: for(int row = 1; row <=3; row++)
5:   for(int col = 1; col <=2 ; col++) {
6:     if(row * col % 2 == 0) continue ROW_LOOP;
7:     count++;
8:   }
9: System.out.println(count);
```

A. 1
B. 2
C. 3
D. 4
E. 6
F. The code will not compile because of line 6.

Self Test 

Self Test (13)

What is the output of the following code snippet?

```
3: boolean keepGoing = true;
4: int result = 15, i = 10;
5: do {
6:   i--;
7:   if(i==8) keepGoing = false;
8:   result -= 2;
9: } while(keepGoing);
10: System.out.println(result);
```

A. 7
B. 9
C. 10
D. 11
E. 15
F. The code will not compile because of line 8.

Self Test 

Self Test (14)

What is the output of the following code snippet?

```
3: do {
4:   int y = 1;
5:   System.out.print(y++ + " ");
6: } while(y <= 10);
```

A. 1 2 3 4 5 6 7 8 9
B. 1 2 3 4 5 6 7 8 9 10
C. 1 2 3 4 5 6 7 8 9 10 11
D. The code will not compile because of line 6.
E. The code contains an infinite loop and does not terminate.

Self Test 

Self Test (15)

What is the output of the following code snippet?

```
3: int x = 1, y = 15;
4: while x < 10
5:   y--;
6:   x++;
7: System.out.println(x+", "+y);
```

A. `10, 5`

B. `10, 6`

C. `11, 5`

D. The code will not compile because of line 3.

E. The code will not compile because of line 4.

F. The code contains an infinite loop and does not terminate.

Self Test 

Self Test (16)

What is the output of the following code snippet?

```
3: int c = 7;
4: int result = 4;
5: result += ++c;
6: System.out.println(result);
```

A. `8`

B. `11`

C. `12`

D. `15`

E. `16`

F. The code will not compile because of line 5.

Self Test 

Self Test (17)

What is the output of the following code snippet?

```
3: int x1 = 50, x2 = 75;
4: boolean b = x1 >= x2;
5: if(b = true) System.out.println("Success");
6: else System.out.println("Failure");
```

A. `Success`

B. `Failure`

C. The code will not compile because of line 4.

D. The code will not compile because of line 5.

Self Test 

Self Test (18)

What is the output of the following code snippet?

```
3: int x = 0;
4: String s = null;
5: if(x == s) System.out.println("Success");
6: else System.out.println("Failure");
```

A. `Success`

B. `Failure`

C. The code will not compile because of line 4.

D. The code will not compile because of line 5.

Self Test 

Self Test (19)

What is the output of the following code?

```
1: public class ArithmeticSample {
2:   public static void main(String[] args) {
3:     int x = 5 * 4 % 3;
4:     System.out.println(x);
5: }}
```

A. 2

B. 3

C. 5

D. 6

E. The code will not compile because of line 3.

Self Test 

Self Test (20)

What is the output of the following code?

```
3: byte a = 40, b = 50;
4: byte sum = (byte) a + b;
5: System.out.println(sum);
```

A. 40

B. 50

C. 90

D. The code will not compile because of line 4.

E. An undefined value.

Self Test 

Self Test (21)

How many times will the following code print "Hello World"?

```
3: for(int i=0; i<10 ; ) {
4:   i = i++;
5:   System.out.println("Hello World");
6: }
```

A. 9
B. 10
C. 11
D. The code will not compile because of line 3.
E. The code will not compile because of line 5.
F. The code contains an infinite loop and does not terminate.

Self Test 

Self Test (22)

What is the output of the following code snippet?

```
3: boolean x = true, z = true;
4: int y = 20;
5: x = (y != 10) ^ (z=false);
6: System.out.println(x+", "+y+", "+z);
```

A. `true, 10, true`
B. `true, 20, false`
C. `false, 20, true`
D. `false, 20, false`
E. `false, 20, true`
F. The code will not compile because of line 5.

Self Test 

Self Test (23)

What is the output of the following code?

```
1: public class TernaryTester {
2:   public static void main(String[] args) {
3:     int x = 5;
4:     System.out.println(x > 2 ? x < 4 ? 10 : 8 : 7);
5: }}
```

A. 5

B. 4

C. 10

D. 8

E. 7

F. The code will not compile because of line 4.

Self Test 

Self Test (24)

What is the output of the following code snippet?

```
3: int x = 4;
4: long y = x * 4 - x++;
5: if(y<10) System.out.println("Too Low");
6: else System.out.println("Just right");
7: else System.out.println("Too High");
```

A. `Too Low`

B. `Just Right`

C. `Too High`

D. Compiles but throws a `NullPointerException`.

E. The code will not compile because of line 6.

F. The code will not compile because of line 7.

Self Test 

Self Test (25)

What is the output of the following code snippet?

```
3: java.util.List<Integer> list = new java.util.ArrayList<Integer>();
4: list.add(10);
5: list.add(14);
6: for(int x : list) {
7:   System.out.print(x + ", ");
8:   break;
9: }
```

A. `10, 14,`

B. `10, 14`

C. `10,`

D. The code will not compile because of line 7.

E. The code will not compile because of line 8.

F. The code contains an infinite loop and does not terminate.

Self Test 

Self Test (26)

What change would allow the following code snippet to compile? (Choose all that apply)

```
3: long x = 10;
4: int y = 2 * x;
```

A. No change; it compiles as is.

B. Cast `x` on line 4 to `int`.

C. Change the data type of `x` on line 3 to `short`.

D. Cast `2 * x` on line 4 to `int`.

E. Change the data type of `y` on line 4 to `short`.

F. Change the data type of `y` on line 4 to `long`.

Self Test 

Self Test (27)

What is the output of the following application?

```
1: public class CompareValues {
2:   public static void main(String[] args) {
3:     int x = 0;
4:     while(x++ < 10) {}
5:     String message = x > 10 ? "Greater than" : false;
6:     System.out.println(message+","+x);
7:   }
8: }
```

A. `Greater than,10`

B. `false,10`

C. `Greater than,11`

D. `false,11`

E. The code will not compile because of line 4.

F. The code will not compile because of line 5.

Self Test 

Self Test (28)

What data type (or types) will allow the following code snippet to compile? (Choose all that apply)

```
byte x = 5;
byte y = 10;
_____ z = x + y;
```

A. `int`

B. `long`

C. `boolean`

D. `double`

E. `short`

F. `byte`

Self Test (29)

Which of the following Java operators can be used with `boolean` variables? (Choose all that apply)

A. `==`

B. `+`

C. `--`

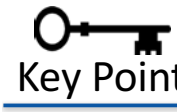
D. `!`

E. `%`

F. `<=`

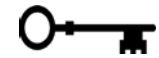
OCA 8 Exam Prep

Section 3:
Core Java APIs



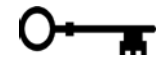
String and StringBuilder

- `String` objects are immutable, and `String` reference variables are not.
- If you create a new `String` without assigning it, it will be lost to your program.
- If you redirect a `String` reference to a new `String`, the old `String` can be lost.
- `String` methods use zero-based indexes, except for the second argument of `substring()`.
- The `String` class is `final`—it cannot be extended.
- When the JVM finds a `String` literal, it is added to the `String` literal pool.
- Strings have a *method* called `length()`—arrays have an *attribute* named `length`.
- `StringBuilder` objects are mutable—they can change without creating a new object.
- `StringBuilder` methods act on the invoking object, and objects can change without an explicit assignment in the statement.
- Remember that chained methods are evaluated from left to right.
- `String` methods to remember: `charAt()`, `concat()`, `equalsIgnoreCase()`, `length()`, `replace()`, `substring()`, `toLowerCase()`, `toString()`, `toUpperCase()`, and `trim()`.
- `StringBuilder` methods to remember: `append()`, `delete()`, `insert()`, `reverse()`, and `toString()`.



Array Declarations

- Arrays can hold primitives or objects, but the array itself is always an object.
- When you declare an array, the brackets can be to the left or to the right of the variable name.
- It is never legal to include the size of an array in the declaration.
- An array of objects can hold any object that passes the IS-A (or instanceof) test for the declared type of the array. For example, if Horse extends Animal, then a Horse object can go into an Animal array.



Arrays (1)

- Arrays can hold primitives or objects, but the array itself is always an object.
- When you declare an array, the brackets can be to the left or right of the name.
- It is never legal to include the size of an array in the declaration.
- You must include the size of an array when you construct it (using `new`) unless you are creating an anonymous array.
- Elements in an array of objects are not automatically created, although primitive array elements are given default values.
- You'll get a `NullPointerException` if you try to use an array element in an object array, if that element does not refer to a real object.

Key Points

Arrays (2)

- Arrays are indexed beginning with zero.
- An `ArrayIndexOutOfBoundsException` occurs if you use a bad index value.
- Arrays have a `length` attribute whose value is the number of array elements.
- The last index you can access is always one less than the length of the array.
- Multidimensional arrays are just arrays of arrays.
- The dimensions in a multidimensional array can have different lengths.
- An array of primitives can accept any value that can be promoted implicitly to the array's declared type—for example, a `byte` variable can go in an `int` array.
- An array of objects can hold any object that passes the IS-A (or `instanceof`) test for the declared type of the array. For example, if `Horse` extends `Animal`, then a `Horse` object can go into an `Animal` array.
- If you assign an array to a previously declared array reference, the array you're assigning must be the same dimension as the reference you're assigning it to.
- You can assign an array of one type to a previously declared array reference of one of its supertypes. For example, a `Honda` array can be assigned to an array declared as type `Car` (assuming `Honda` extends `Car`).

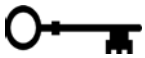
Key Points

ArrayList

- `ArrayLists` allow you to resize your list and make insertions and deletions to your list far more easily than arrays.
- For the OCA 7 exam, the only `ArrayList` declarations you need to know are of this form:

```
ArrayList<type> myList = new ArrayList<type>();
List<type> myList2 = new ArrayList<type>();  // polymorphic
```

- `ArrayLists` can hold only objects, not primitives, but remember that autoboxing can make it look like you're adding primitives to an `ArrayList` when in fact you're adding a wrapper version of a primitive.
- An `ArrayList`'s index starts at 0.
- `ArrayLists` can have duplicate entries. Note: Determining whether two objects are duplicates is trickier than it seems and doesn't come up until the OCP 7 exam.
- `ArrayList` methods to remember: `add(element)`, `add(index, element)`, `clear()`, `contains()`, `get(index)`, `indexOf()`, `remove(index)`, `remove(object)`, and `size()`.



Dates & Times

- LocalDate contains just a date
- LocalTime contains just a time
- LocalDateTime contains
- All have private constructors
- Created using ClassName.now() or ClassName.of()
- Manipulated using plusXXX or minusXXX methods
- Period class = number of days/months/years
- Add/subtract Period from/to LocalDate/LocalDateTime
- DateTimeFormatter is used to format and output
- Date and time classes are immutable so use return value

Self Test (1)

What is the output of the following code?

```
LocalDate date = LocalDate.parse("2018-04-30", DateTimeFormatter.ISO_LOCAL_
DATE);
date.plusDays(2);
date.plusHours(3);
System.out.println(date.getYear() + " " + date.getMonth() + " "
+ date.getDayOfMonth());
```

A. `2018 APRIL 2`

B. `2018 APRIL 30`

C. `2018 MAY 2`

D. The code does not compile.

E. A runtime exception is thrown.

Self Test 

Self Test (2)

Which are the results of the following code? (Choose all that apply)

```
String letters = "abcdef";
System.out.println(letters.length());
System.out.println(letters.charAt(3));
System.out.println(letters.charAt(6));
```

A. 5

B. 6

C. c

D. d

E. An exception is thrown.

F. The code does not compile.

Self Test 

Self Test (3)

What is the result of the following code?

```
2: String s1 = "java";
3: StringBuilder s2 = new StringBuilder("java");
4: if (s1 == s2)
5:  System.out.print("1");
6: if (s1.equals(s2))
7:  System.out.print("2");
```

A. 1

B. 2

C. 12

D. No output is printed.

E. An exception is thrown.

F. The code does not compile.

Self Test 

Self Test (4)

Which of the following are output by this code? (Choose all that apply)

```
3: String s = "Hello";
4: String t = new String(s);
5: if ("Hello".equals(s)) System.out.println("one");
6: if (t == s) System.out.println("two");
7: if (t.equals(s)) System.out.println("three");
8: if ("Hello" == s) System.out.println("four");
9: if ("Hello" == t) System.out.println("five");
```

A. one

B. two

C. three

D. four

E. five

F. The code does not compile.

Self Test 

Self Test (5)

Given:

```
public class Mutant {
  public static void main(String[] args) {
    StringBuilder sb = new StringBuilder("abc");
    String s = "abc";
    sb.reverse().append("d");
    s.toUpperCase().concat("d");
    System.out.println("." + sb + ". ." + s + ".");
  }
}
```

Which two substrings will be included in the result? (Choose two.)

A. .abc.

B. .ABCd.

C. .ABCD.

D. .cbad.

E. .dcba.

Self Test 

Self Test (6)

What is the output of the following code?

```
LocalDate date = LocalDate.of(2018, Month.APRIL, 30);
date.plusDays(2);
date.plusYears(3);
System.out.println(date.getYear() + " " + date.getMonth() + " "
+ date.getDayOfMonth());
```

A. `2018 APRIL 2`

B. `2018 APRIL 30`

C. `2018 MAY 2`

D. `2021 APRIL 2`

E. `2021 APRIL 30`

F. `2021 MAY 2`

G. A runtime exception is thrown.

Self Test 

Self Test (7)

Given:

```
public class Hilltop {
  public static void main(String[] args) {
    String[] horses = new String[5];
    horses[4] = null;
    for(int i = 0; i < horses.length; i++) {
      if(i < args.length)
        horses[i] = args[i];
      System.out.print(horses[i].toUpperCase() + " ");
    }
  }
}
```

And, if the code compiles, the command line:

```
java Hilltop eyra vafi draumur kara
```

What is the result?

A. `EYRA VAFI DRAUMUR KARA`

B. `EYRA VAFI DRAUMUR KARA null`

C. An exception is thrown with no other output

D. `EYRA VAFI DRAUMUR KARA`, and then a `NullPointerException`

E. `EYRA VAFI DRAUMUR KARA`, and then an `ArrayIndexOutOfBoundsException`

F. Compilation fails

Self Test 

Self Test (8)

What is the result of the following code?

```
public class Lion {
  public void roar(String roar1, StringBuilder roar2) {
    roar1.concat("!!!");
    roar2.append("!!!");
  }
public static void main(String[] args) {
  String roar1 = "roar";
  StringBuilder roar2 = new StringBuilder("roar");
  new Lion().roar(roar1, roar2);
  System.out.println(roar1 + " " + roar2);
} }
```

A. `roar roar`

B. `roar roar!!!`

C. `roar!!! roar`

D. `roar!!! roar!!!`

E. An exception is thrown.

F. The code does not compile.

Self Test 

Self Test (9)

Given:

```
public class Actors {
  public static void main(String[] args) {
    char[] ca = {0x4e, \u004e, 78};
    System.out.println((ca[0] == ca[1]) + " " + (ca[0] == ca[2]));
  }
}
```

What is the result?

A. `true true`

B. `true false`

C. `false true`

D. `false false`

E. Compilation fails

Self Test 

Self Test (10)

Which are true statements? (Choose all that apply)

A. An immutable object can be modified.

B. An immutable object cannot be modified.

C. An immutable object can be garbage collected.

D. An immutable object cannot be garbage collected.

E. `String` is immutable.

F. `StringBuffer` is immutable.

G. `StringBuilder` is immutable.

Self Test 

Self Test (11)

What is the output of the following code?

```
LocalDate date = LocalDate.of(2018, Month.APRIL, 40);
System.out.println(date.getYear() + " " + date.getMonth() + " "
+ date.getDayOfMonth());
```

A. `2018 APRIL 4`

B. `2018 APRIL 30`

C. `2018 MAY 10`

D. Another date.

E. The code does not compile.

F. A runtime exception is thrown.

Self Test 

Self Test (12)

Given:

```
1. class Dims {
2.   public static void main(String[] args) {
3.     int[][] a = {{1,2}, {3,4}};
4.     int[] b = (int[]) a[1];
5.     Object o1 = a;
6.     int[][] a2 = (int[][]) o1;
7.     int[] b2 = (int[]) o1;
8.     System.out.println(b[1]);
9. } }
```

What is the result? (Choose all that apply.)

A. 2
B. 4
C. An exception is thrown at runtime
D. Compilation fails due to an error on line 4
E. Compilation fails due to an error on line 5
F. Compilation fails due to an error on line 6
G. Compilation fails due to an error on line 7

Self Test 

Self Test (13)

What is the output of the following code?

```
LocalDateTime d = LocalDateTime.of(2015, 5, 10, 11, 22, 33);
Period p = Period.ofDays(1).ofYears(2);
d = d.minus(p);
DateTimeFormatter f = DateTimeFormatter.ofLocalizedDateTime(FormatStyle
.SHORT);
System.out.println(f.format(d));
```

A. 5/9/13 11:22 AM
B. 5/10/13 11:22 AM
C. 5/9/14
D. 5/10/14
E. The code does not compile.
F. A runtime exception is thrown.

Self Test 

Self Test (14)

Given:

```
import java.util.*;
public class Sequence {
  public static void main(String[] args) {
    ArrayList<String> myList = new ArrayList<String>();
    myList.add("apple");
    myList.add("carrot");
    myList.add("banana");
    myList.add(1, "plum");
    System.out.print(myList);
  }
}
```

What is the result?

A. `[apple, banana, carrot, plum]`
B. `[apple, plum, carrot, banana]`
C. `[apple, plum, banana, carrot]`
D. `[plum, banana, carrot, apple]`
E. `[plum, apple, carrot, banana]`
F. `[banana, plum, carrot, apple]`
G. Compilation fails

Self Test 

Self Test (15)

What is the result of the following?

```
List<String> one = new ArrayList<String>();
one.add("abc");
List<String> two = new ArrayList<>();
two.add("abc");
if (one == two)
  System.out.println("A");
else if (one.equals(two))
  System.out.println("B");
else
  System.out.println("C");
```

A. `A`
B. `B`
C. `C`
D. An exception is thrown.
E. The code does not compile.

Self Test 

Self Test (16)

Given:

```
public class Tailor {
  public static void main(String[] args) {
    byte[][] ba = {{1,2,3,4}, {1,2,3}};
    System.out.println(ba[1].length + " " + ba.length);
  }
}
```

What is the result?

A. 2 4
B. 2 7
C. 3 2
D. 3 7
E. 4 2
F. 4 7
G. Compilation fails

Self Test 

Self Test (17)

What is the result of the following code?

```
7: StringBuilder sb = new StringBuilder();
8: sb.append("aaa").insert(1, "bb").insert(4, "ccc");
9: System.out.println(sb);
```

A. abbaaccc
B. abbaccca
C. bbaaaccc
D. bbaaccca
E. An exception is thrown.
F. The code does not compile.

Self Test 

Self Test (18)

Given:

```
3. public class Theory {
4.   public static void main(String[] args) {
5.     String s1 = "abc";
6.     String s2 = s1;
7.     s1 += "d";
8.     System.out.println(s1 + " " + s2 + " " + (s1==s2));
9.
10.    StringBuilder sb1 = new StringBuilder("abc");
11.    StringBuilder sb2 = sb1;
12.    sb1.append("d");
13.    System.out.println(sb1 + " " + sb2 + " " + (sb1==sb2));
14.  }
15. }
```

Which are true? (Choose all that apply.)

A. Compilation fails
B. The first line of output is `abc abc true`
C. The first line of output is `abc abc false`
D. The first line of output is `abcd abc false`
E. The second line of output is `abcd abc false`
F. The second line of output is `abcd abcd true`
G. The second line of output is `abcd abcd false`

Self Test 

Self Test (19)

What is the output of the following code?

```
LocalDateTime d = LocalDateTime.of(2015, 5, 10, 11, 22, 33);
Period p = Period.of(1, 2, 3);
d = d.minus(p);
DateTimeFormatter f = DateTimeFormatter.ofLocalizedTime(FormatStyle.SHORT);
System.out.println(d.format(f));
```

A. `3/7/14 11:22 AM`
B. `5/10/15 11:22 AM`
C. `3/7/14`
D. `5/10/15`
E. `11:22 AM`
F. The code does not compile.
G. A runtime exception is thrown.

Self Test 

Self Test (20)

Given:

```
3. class Box {
 4.   int size;
 5.   Box(int s) { size = s; }
 6. }
 7. public class Laser {
 8.   public static void main(String[] args) {
 9.     Box b1 = new Box(5);
10.     Box[] ba = go(b1, new Box(6));
11.     ba[0] = b1;
12.    for(Box b : ba) System.out.print(b.size + " ");
13.   }
14.   static Box[] go(Box b1, Box b2) {
15.     b1.size = 4;
16.     Box[] ma = {b2, b1};
17.     return ma;
18.   }
19. }
```

What is the result?

A. `4 4`
B. `5 4`
C. `6 4`
D. `4 5`
E. `5 5`
F. Compilation fails

Self Test 

Self Test (21)

Which of the following can be inserted into the blank to create a date of June 21, 2014? (Choose all that apply)

```
import java.time.*;

public class StartOfSummer {

  public static void main(String[] args) {
  LocalDate date = ____________________________
  }

}
```

A. `new LocalDate(2014, 5, 21);`
B. `new LocalDate(2014, 6, 21);`
C. `LocalDate.of(2014, 5, 21);`
D. `LocalDate.of(2014, 6, 21);`
E. `LocalDate.of(2014, Calendar.JUNE, 21);`
F. `LocalDate.of(2014, Month.JUNE, 21);`

Self Test 

Self Test (22)

Given:

```
public class Hedges {
  public static void main(String[] args) {
    String s = "JAVA";
    s = s + "rocks";
    s = s.substring(4,8);
    s.toUpperCase();
    System.out.println(s);
  }
}
```

What is the result?

A. JAVA
B. JAVAROCKS
C. rocks
D. rock
E. ROCKS
F. ROCK
G. Compilation fails

Self Test 

Self Test (23)

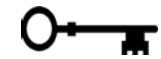
What is output by the following code? (Choose all that apply)

```
1: public class Fish {
2:  public static void main(String[] args) {
3:    int numFish = 4;
4:    String fishType = "tuna";
5:    String anotherFish = numFish + 1;
6:    System.out.println(anotherFish + " " + fishType);
7:    System.out.println(numFish + " " + 1);
8:  } }
```

A. 4 1
B. 41
C. 5
D. 5 tuna
E. 5tuna
F. 51tuna
G. The code does not compile.

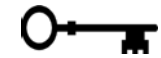
OCA 8 Exam Prep

Section 4:
Methods and Encapsulation



Encapsulation

- Encapsulation helps hide implementation behind an interface (or API).
- Encapsulated code has two features:
  - Instance variables are kept protected (usually with the `private` modifier).
  - Getter and setter methods provide access to instance variables.
- IS-A refers to inheritance or implementation.
- IS-A is expressed with the keyword `extends` or `implements`.
- IS-A, "inherits from," and "is a subtype of" are all equivalent expressions.
- HAS-A means an instance of one class "has a" reference to an instance of another class or another instance of the same class.



Statics

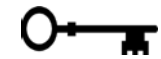
- Use `static` methods to implement behaviors that are not affected by the state of any instances.
- Use `static` variables to hold data that is class specific as opposed to instance specific—there will be only one copy of a `static` variable.
- All `static` members belong to the class, not to any instance.
- A `static` method can't access an instance variable directly.
- Use the dot operator to access `static` members, but remember that using a reference variable with the dot operator is really a syntax trick, and the compiler will substitute the class name for the reference variable; for instance:

```
d.doStuff();
```

  becomes

```
Dog.doStuff();
```

- `static` methods can't be overridden, but they can be redefined.



Initialization Blocks

- Use `static init` blocks—`static { /* code here */ }`—for code you want to have run once, when the class is first loaded. Multiple blocks run from the top down.
- Use normal `init` blocks—`{ /* code here }`—for code you want to have run for every new instance, right after all the super constructors have run. Again, multiple blocks run from the top of the class down.

Constructors and Instantiation (1)

- A constructor is always invoked when a new object is created.
- Each superclass in an object's inheritance tree will have a constructor called.
- Every class, even an abstract class, has at least one constructor.
- Constructors must have the same name as the class.
- Constructors don't have a return type. If you see code with a return type, it's a method with the same name as the class; it's not a constructor.
- Typical constructor execution occurs as follows:
  - The constructor calls its superclass constructor, which calls its superclass constructor, and so on all the way up to the `Object` constructor.
  - The `Object` constructor executes and then returns to the calling constructor, which runs to completion and then returns to its calling constructor, and so on back down to the completion of the constructor of the actual instance being created.
- Constructors can use any access modifier (even `private`!).

Constructors and Instantiation (2)

- The compiler will create a default constructor if you don't create any constructors in your class.
- The default constructor is a no-arg constructor with a no-arg call to `super()`.
- The first statement of every constructor must be a call either to `this()` (an overloaded constructor) or to `super()`.
- The compiler will add a call to `super()` unless you have already put in a call to `this()` or `super()`.
- Instance members are accessible only after the `super` constructor runs.
- `Abstract` classes have constructors that are called when a concrete subclass is instantiated.
- Interfaces do not have constructors.
- If your superclass does not have a no-arg constructor, you must create a constructor and insert a call to `super()` with arguments matching those of the superclass constructor.

Constructors and Instantiation (3)

- Constructors are never inherited; thus they cannot be overridden.
- A constructor can be directly invoked only by another constructor (using a call to `super()` or `this()`).
- Regarding issues with calls to `this()`:
  - They may appear only as the first statement in a constructor.
  - The argument list determines which overloaded constructor is called.
  - Constructors can call constructors, and so on, but sooner or later one of them better call `super()` or the stack will explode.
  - Calls to `this()` and `super()` cannot be in the same constructor. You can have one or the other, but never both.

Return Types

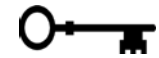
- Overloaded methods can change return types; overridden methods cannot, except in the case of covariant returns.
- Object reference return types can accept `null` as a return value.
- An array is a legal return type, both to declare and return as a value.
- For methods with primitive return types, any value that can be implicitly converted to the return type can be returned.
- Nothing can be returned from a `void`, but you can return nothing. You're allowed to simply say `return` in any method with a `void` return type to bust out of a method early. But you can't return nothing from a method with a non-`void` return type.
- Methods with an object reference return type can return a subtype.
- Methods with an interface return type can return any implementer.

Overriding and Overloading (1)

- Methods can be overridden or overloaded; constructors can be overloaded but not overridden.
- With respect to the method it overrides, the overriding method
  - Must have the same argument list
  - Must have the same return type, except that, as of Java 5, the return type can be a subclass, and this is known as a covariant return
  - Must not have a more restrictive access modifier
  - May have a less restrictive access modifier
  - Must not throw new or broader checked exceptions
  - May throw fewer or narrower checked exceptions, or any unchecked exception
- `final` methods cannot be overridden.
- Only inherited methods may be overridden, and remember that private methods are not inherited.

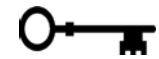
Overriding and Overloading (2)

- A subclass uses `super.overriddenMethodName()` to call the superclass version of an overridden method.
- Overloading means reusing a method name but with different arguments.
- Overloaded methods
  - Must have different argument lists
  - May have different return types, if argument lists are also different
  - May have different access modifiers
  - May throw different exceptions
- Methods from a superclass can be overloaded in a subclass.
- Polymorphism applies to overriding, not to overloading.
- Object type (not the reference variable's type) determines which overridden method is used at runtime.
- Reference type determines which overloaded method will be used at compile time.



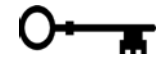
Method Structure

- start with an access modifier of:
  - public (use from any package)
  - private (use within same class)
  - protected (use within same package or subclasses)
  - blank a.k.a. default access (use within same package)
- followed by an optional specifier:
  - static
  - final
  - abstract
- then the return type: void or any Java type
- then the method name follows (standard Java identifier rules0
- zero or more parameters in parentheses
- zero of more exception types, comma separated
- zero or more statements in braces



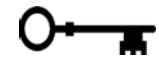
Member Access Modifiers (1)

- Methods and instance (non-local) variables are known as “members”
- Members can be modified by all 4 access modifiers – public, private, protected and default
- If visibility allows, code in one class can access members in another class and inherit members from a superclass
- If a class cannot be accessed, neither can its members
- EXAM TIP: Determine class visibility before determining member visibility



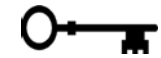
Member Access Modifiers (2)

- public members can be accessed anywhere the class can
- If a superclass member is public the subclass inherits regardless of package
- Members accessed without the . operator must be in the same class
- this always refers to the currently executing object
- this.method() as the same as method()
- private members can only be accessed by code in the same class



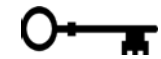
Member Access Modifiers (3)

- private members are not visible to subclasses so cannot be inherited
- protected and default only vary for inheritance
- subclasses can inherit protected members but can’t access them in an instance of the superclass
- protected members are not accessible to other classes in the same package as the subclass



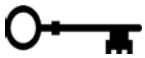
Other Member Modifiers (1)

- final methods cannot be overridden
- abstract methods must have a signature and a return type and can have a throws clause but no implementation (use ; in place of {})
- Code in the curly braces is not mandatory
- Methods with curly braces are not abstract
- The first concrete class inheriting from an abstract class must implement all the abstract methods



Other Member Modifiers (2)

- The synchronized modifier applies only to methods and code blocks.
- synchronized methods can have any access control and can also be marked final.
- abstract methods must be implemented by a subclass, so they must be
- inheritable. For that reason:
- abstract methods cannot be private.
- abstract methods cannot be final.
- The native modifier applies only to methods.
- The strictfp modifier applies only to classes and methods.



Lambda Expressions

- allow passing around blocks of code
- syntax (String a, String b) -> { return a.equals(b); }
- parameter types can be omitted
- with one parameter and no type, the parentheses are optional
- braces and return statement can be omitted for a single statement
- short form a -> a.equals(b)
- lambdas passed as a parameter to a method expecting a functional interface
- Predicate is a common interface
- Predicate's method is boolean test(T t) that returns a boolean and takes any type (T)
- the removeIf() method on ArrayList takes a Predicate

Self Test (1)

What is the result of the following class?

```
1: import java.util.function.*;
2:
3: public class Panda {
4:   int age;
5:   public static void main(String[] args) {
6:     Panda p1 = new Panda();
7:     p1.age = 1;
8:     check(p1, p -> p.age < 5);
9:   }
10:   private static void check(Panda panda, Predicate<Panda> pred) {
11:     String result = pred.test(panda) ? "match" : "not match";
12:     System.out.print(result);
13: } }
```

A. `match`

B. `not match`

C. Compiler error on line 8.

D. Compiler error on line 10.

E. Compiler error on line 11.

F. A runtime exception is thrown.

Self Test 

Self Test (2)

What is the result of the following?

```
1: public class Order {
2:   String value = "t";
3:   { value += "a"; }
4:   { value += "c"; }
5:   public Order() {
6:     value += "b";
7:   }
8:   public Order(String s) {
9:     value += s;
10:  }
11:  public static void main(String[] args) {
12:    Order order = new Order("f");
13:    order = new Order();
14:    System.out.println(order.value);
15:  } }
```

A. tacb
B. tacf
C. tacbf
D. tacfb
E. tacftacb
F. The code does not compile.
G. An exception is thrown.

Self Test 

Self Test (3)

Which of the following complete the constructor so that this code prints out `50`? (Choose all that apply)

```
public class Cheetah {
  int numSpots;
  public Cheetah(int numSpots) {
    // INSERT CODE HERE
  }
  public static void main(String[] args) {
    System.out.println(new Cheetah(50).numSpots);
  }
}
```

A. `numSpots = numSpots;`
B. `numSpots = this.numSpots;`
C. `this.numSpots = numSpots;`
D. `numSpots = super.numSpots;`
E. `super.numSpots = numSpots;`
F. None of the above.

Self Test

Self Test (4)

Which code can be inserted to have the code print 2?

```
public class BirdSeed {
  private int numberBags;
  boolean call;

  public BirdSeed() {
    // LINE 1
    call = false;
    // LINE 2
  }
  public BirdSeed(int numberBags) {
    this.numberBags = numberBags;
  }
  public static void main(String[] args) {
    BirdSeed seed = new BirdSeed();
    System.out.println(seed.numberBags);
  } }
```

A. Replace line 1 with `BirdSeed(2);`
B. Replace line 2 with `BirdSeed(2);`
C. Replace line 1 with `new BirdSeed(2);`
D. Replace line 2 with `new BirdSeed(2);`
E. Replace line 1 with `this(2);`
F. Replace line 2 with `this(2);`

Self Test

Self Test (5)

Which of the following are true about the following code? (Choose all that apply)

```
public class Create {
  Create() {
    System.out.print("1 ");
  }
  Create(int num) {
    System.out.print("2 ");
  }
  Create(Integer num) {
    System.out.print("3 ");
  }
  Create(Object num) {
    System.out.print("4 ");
  }
  Create(int... nums) {
    System.out.print("5 ");
  }
  public static void main(String[] args) {
    new Create(100);
    new Create(1000L);
  }
}
```

A. The code prints out `2 4`.
B. The code prints out `3 4`.
C. The code prints out `4 2`.
D. The code prints out `4 4`.
E. The code prints `3 4` if you remove the constructor `Create(int num)`.
F. The code prints `4 4` if you remove the constructor `Create(int num)`.
G. The code prints `5 4` if you remove the constructor `Create(int num)`.

Self Test 

Self Test (6)

Given the following `my.school.ClassRoom` and `my.city.School` class definitions, which line numbers in `main()` generate a compiler error? (Choose all that apply)

```
1: package my.school;
2: public class Classroom {
3:   private int roomNumber;
4:   protected String teacherName;
5:   static int globalKey = 54321;
6:   public int floor = 3;
7:   Classroom(int r, String t) {
8:     roomNumber = r;
9:     teacherName = t; } }
```

```
1: package my.city;
2: import my.school.*;
3: public class School {
4:   public static void main(String[] args) {
5:     System.out.println(Classroom.globalKey);
6:     Classroom room = new Classroom(101, ""Mrs. Anderson");
7:     System.out.println(room.roomNumber);
8:     System.out.println(room.floor);
9:     System.out.println(room.teacherName); } }
```

A. None, the code compiles fine.

B. Line 5

C. Line 6

D. Line 7

E. Line 8

F. Line 9

Self Test 

Self Test (7)

What is the result of the following program?

```
1: public class Squares {
2:   public static long square(int x) {
3:     long y = x * (long) x;
4:     x = -1;
5:     return y;
6:   }
7:   public static void main(String[] args) {
8:     int value = 9;
9:     long result = square(value);
10:     System.out.println(value);
11:   } }
```

A. `-1`

B. `9`

C. `81`

D. Compiler error on line 9.

E. Compiler error on a different line.

Self Test 

Self Test (8)

Which of the following are true? (Choose 2)

A. `this()` can be called from anywhere in a constructor.

B. `this()` can be called from any instance method in the class.

C. `this.variableName` can be called from any instance method in the class.

D. `this.variableName` can be called from any static method in the class.

E. You must include a default constructor in the code if the compiler does not include one.

F. You can call the default constructor written by the compiler using `this()`.

G. You can access a private constructor with the `main()` method.

Self Test 

Self Test (9)

Which lambda can replace the `MySecret` class to return the same value? (Choose all that apply)

```
interface Secret {
  String magic(double d);
}

class MySecret implements Secret {
  public String magic(double d) {
    return "Poof";
  }
}
```

A. `caller((e) -> "Poof");`

B. `caller((e) -> {"Poof"});`

C. `caller((e) -> { String e = ""; "Poof" });`

D. `caller((e) -> { String e = ""; return "Poof"; });`

E. `caller((e) -> { String e = ""; return "Poof" });`

F. `caller((e) -> { String f = ""; return "Poof"; });`

Self Test 

Self Test (10)

Which of these classes compile and use a default constructor? (Choose all that apply)

A. `public class Bird { }`

B. `public class Bird { public bird() {} }`

C. `public class Bird { public bird(String name) {} }`

D. `public class Bird { public Bird() {} }`

E. `public class Bird { Bird(String name) {} }`

F. `public class Bird { private Bird(int age) {} }`

G. `public class Bird { void Bird() { } }`

Self Test 

Self Test (11)

Given the following method, which of the method calls return 2? (Choose all that apply)

```
public int howMany(boolean b, boolean... b2) {
  return b2.length;
}
```

A. `howMany();`

B. `howMany(true);`

C. `howMany(true, true);`

D. `howMany(true, true, true);`

E. `howMany(true, {true});`

F. `howMany(true, {true, true});`

G. `howMany(true, new boolean[2]);`

Self Test 

Self Test (12)

Which of the following are output by the following code? (Choose all that apply)

```
public class StringBuilders {
  public static StringBuilder work(StringBuilder a,
StringBuilder b) {
    a = new StringBuilder("a");
    b.append("b");
    return a;
  }
  public static void main(String[] args) {
    StringBuilder s1 = new StringBuilder("s1");
    StringBuilder s2 = new StringBuilder("s2");
    StringBuilder s3 = work(s1, s2);
    System.out.println("s1 = " + s1);
    System.out.println("s2 = " + s2);
    System.out.println("s3 = " + s3);
  }
}
```

A. `s1 = a`

B. `s1 = s1`

C. `s2 = s2`

D. `s2 = s2b`

E. `s3 = a`

F. `s3 = null`

G. The code does not compile.

Self Test 

Self Test (13)

Which are methods using JavaBeans naming conventions for accessors and mutators? (Choose all that apply)

A. `public boolean getCanSwim() { return canSwim;}`

B. `public boolean canSwim() { return numberWings;}`

C. `public int getNumWings() { return numberWings;}`

D. `public int numWings() { return numberWings;}`

E. `public void setCanSwim(boolean b) { canSwim = b;}`

Self Test 

Self Test (14)

What is the result of the following statements?

```
1:  public class Test {
2:    public void print(byte x) {
3:      System.out.print("byte");
4:    }
5:    public void print(int x) {
6:      System.out.print("int");
7:    }
8:    public void print(float x) {
9:      System.out.print("float");
10:   }
11:   public void print(Object x) {
12:     System.out.print("Object");
13:   }
14:   public static void main(String[] args) {
15:     Test t = new Test();
16:     short s = 123;
17:     t.print(s);
18:     t.print(true);
19:     t.print(6.789);
20:   }
21: }
```

A. `bytefloatObject`
B. `intfloatObject`
C. `byteObjectfloat`
D. `intObjectfloat`
E. `intObjectObject`
F. `byteObjectObject`

Self Test 

Self Test (15)

Which of the following can replace line 2 to make this code compile? (Choose all that apply)

```
1: import java.util.*;
2: // INSERT CODE HERE
3: public class Imports {
4:  public void method(ArrayList<String> list) {
5:    sort(list);
6:  }
7: }
```

A. `import static java.util.Collections;`
B. `import static java.util.Collections.*;`
C. `import static java.util.Collections.sort(ArrayList<String>);`
D. `static import java.util.Collections;`
E. `static import java.util.Collections.*;`
F. `static import java.util.Collections.sort(ArrayList<String>);`

Self Test 

Self Test (16)

How many compiler errors are in the following code?

```
1: public class RopeSwing {
2:   private static final String leftRope;
3:   private static final String rightRope;
4:   private static final String bench;
5:   private static final String name = "name";
6:   static {
7:     leftRope = "left";
8:     rightRope = "right";
9:   }
10:  static {
11:    name = "name";
12:    rightRope = "right";
13:  }
14:  public static void main(String[] args) {
15:    bench = "bench";
16:  }
17: }
```

A. 0
B. 1
C. 2
D. 3
E. 4
F. 5

Self Test 

Self Test (17)

Which of the following compile? (Choose all that apply)

A. `public void moreA(int... nums) {}`
B. `public void moreB(String values, int... nums) {}`
C. `public void moreC(int... nums, String values) {}`
D. `public void moreD(String... values, int... nums) {}`
E. `public void moreE(String[] values, ...int nums) {}`
F. `public void moreF(String... values, int[] nums) {}`
G. `public void moreG(String[] values, int[] nums) {}`

Self Test 

Self Test (18)

What is the output of the following code?

```
import rope.*;
import static rope.Rope.*;
public class RopeSwing {
  private static Rope rope1 = new Rope();
  private static Rope rope2 = new Rope();
  {
    System.out.println(rope1.length);
  }
  public static void main(String[] args) {
    rope1.length = 2;
    rope2.length = 8;
    System.out.println(rope1.length);
  }
}

package rope;
public class Rope {
  public static int length = 0;
}
```

A. 02
B. 08
C. 2
D. 8
E. The code does not compile.
F. An exception is thrown.

Self Test 

Self Test (19)

Which of the following will compile when inserted in the following code? (Choose all that apply)

```
public class Order3 {
  final String value1 = "1";
  static String value2 = "2";
  String value3 = "3";
  {
    // CODE SNIPPET 1
  }
  static {
    // CODE SNIPPET 2
  }
}
```

A. `value1 = "d";` instead of `// CODE SNIPPET 1`
B. `value2 = "e";` instead of `// CODE SNIPPET 1`
C. `value3 = "f";` instead of `// CODE SNIPPET 1`
D. `value1 = "g";` instead of `// CODE SNIPPET 2`
E. `value2 = "h";` instead of `// CODE SNIPPET 2`
F. `value3 = "i";` instead of `// CODE SNIPPET 2`

Self Test 

Self Test (20)

Which are true of the following code? (Choose all that apply)

```
1:  public class Rope {
2:    public static void swing() {
3:      System.out.print("swing ");
4:    }
5:    public void climb() {
6:      System.out.println("climb ");
7:    }
8:    public static void play() {
9:      swing();
10:     climb();
11:   }
12:   public static void main(String[] args) {
13:     Rope rope = new Rope();
14:     rope.play();
15:     Rope rope2 = null;
16:     rope2.play();
17:   }
18: }
```

A. The code compiles as is.
B. There is exactly one compiler error in the code.
C. There are exactly two compiler errors in the code.
D. If the lines with compiler errors are removed, the output is `climb climb`.
E. If the lines with compiler errors are removed, the output is `swing swing`.
F. If the lines with compile errors are removed, the code throws a `NullPointerException`.

Self Test 

Self Test (21)

Which of the following lambda expressions can fill in the blank? (Choose all that apply)

```
List<String> list = new ArrayList<>();
list.removeIf(____________________);
```

A. `s -> s.isEmpty()`
B. `s -> {s.isEmpty()}`
C. `s -> {s.isEmpty();}`
D. `s -> {return s.isEmpty();}`
E. `String s -> s.isEmpty()`
F. `(String s) -> s.isEmpty()`

Self Test 

Self Test (22)

What is the output of the following code?

```
1: package rope;
2: public class Rope {
3:  public static int LENGTH = 5;
4:  static {
5:    LENGTH = 10;
6:  }
7:  public static void swing() {
8:    System.out.print("swing ");
9:  }
10: }
```

```
1: import rope.*;
2: import static rope.Rope.*;
3: public class Chimp {
4:  public static void main(String[] args) {
5:    Rope.swing();
6:    new Rope().swing();
7:    System.out.println(LENGTH);
8:  }
9: }
```

A. `swing swing 5`
B. `swing swing 10`
C. Compiler error on line 2 of `Chimp`.
D. Compiler error on line 5 of `Chimp`.
E. Compiler error on line 6 of `Chimp`.
F. Compiler error on line 7 of `Chimp`.

Self Test 

Self Test (23)

Which of the following are true? (Choose all that apply)

A. Encapsulation uses package private instance variables.
B. Encapsulation uses private instance variables.
C. Encapsulation allows setters.
D. Immutability uses package private instance variables.
E. Immutability uses private instance variables.
F. Immutability allows setters.

Self Test 

Self Test (24)

What is the result of the following code?

```
1: interface Climb {
2:   boolean isTooHigh(int height, int limit);
3: }
4:
5: public class Climber {
6:   public static void main(String[] args) {
7:   check((h, l) -> h.append(l).isEmpty(), 5);
8:   }
9:   private static void check(Climb climb, int height) {
10:    if (climb.isTooHigh(height, 10))
11:      System.out.println("too high");
12:    else
13:      System.out.println("ok");
14:  }
15: }
```

A. `ok`

B. `too high`

C. Compiler error on line 7.

D. Compiler error on line 10.

E. Compiler error on a different line.

F. A runtime exception is thrown.

Self Test 

Self Test (25)

Which of the following are true? (Choose all that apply)

A. Package private access is more lenient than protected access.

B. A `public` class that has private fields and package private methods is not visible to classes outside the package.

C. You can use access modifiers so only some of the classes in a package see a particular package private class.

D. You can use access modifiers to allow read access to all methods, but not any instance variables.

E. You can use access modifiers to restrict read access to all classes that begin with the word `Test`.

Self Test 

Self Test (26)

Which of the following methods compile? (Choose all that apply)

A. `public void methodA() { return;}`

B. `public void methodB() { return null;}`

C. `public void methodD() {}`

D. `public int methodD() { return 9;}`

E. `public int methodE() { return 9.0;}`

F. `public int methodF() { return;}`

G. `public int methodG() { return null;}`

Self Test 

Self Test (27)

Which of the following compile? (Choose all that apply)

A. `final static void method4() { }`

B. `public final int void method() { }`

C. `private void int method() { }`

D. `static final void method3() { }`

E. `void final method() {}`

F. `void public method() { }`

Self Test 

Self Test (28)

What is the result of the following?

```
1:  public class Order {
2:    static String result = "";
3:    { result += "c"; }
4:    static
5:    { result += "u"; }
6:    { result += "r"; }
7: }
```

```
1: public class OrderDriver {
2:   public static void main(String[] args) {
3:     System.out.print(Order.result + " ");
4:     System.out.print(Order.result + " ");
5:     new Order();
6:     new Order();
7:     System.out.print(Order.result + " ");
8:   }
9: }
```

A. `curur`
B. `ucrcr`
C. `u ucrcr`
D. `u u curcur`
E. `u u ucrcr`
F. `ur ur urc`
G. The code does not compile.

Self Test 

Self Test (29)

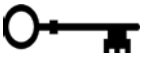
Which of the following can fill in the blank in this code to make it compile? (Choose all that apply)

```
public class Ant {
  _____ void method() { }
}
```

A. `default`
B. `final`
C. `private`
D. `Public`
E. `String`
F. `zzz:`

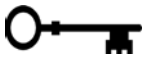
OCA 8 Exam Prep

Section 5:
Class Design


Key Points

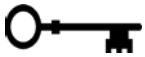
Class Access Modifiers

- There are 3 access modifiers: public, protected and private
- There is a 4th access level: default
- Classes can only have public or default access
- default classes can only be seen in the package
- public classes can be seen everywhere
- If a class can be seen then:
  - an instance can be created and methods/properties used
  - it can be extended (subclassed)



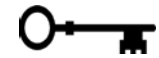
Other Class Modifiers

- Classes can be modified with final, abstract and strictfp
- A class cannot be both final and abstract
- final classes cannot be subclassed
- abstract classes cannot be instantiated
- If a class has an abstract method the class must be abstract
- abstract classes can have both abstract and concrete methods
- The first concrete class to extend an abstract class must implement all the abstract methods



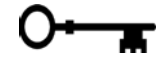
Interface Implementation (1)

- An interface is a contract for what a class can do but says nothing about how to do it
- Interfaces can be implemented by any class from any inheritance tree
- An interface is like a 100% abstract class and is implicitly abstract even if you don't type the word
- Interfaces can only have abstract methods
- Interface methods are by default public and abstract – explicit declaration is optional
- Interfaces can have constants which are implicitly public, static and final – explicit declaration is optional in any combination



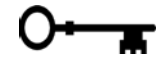
Interface Implementation (2)

- A class implementing an interface can be abstract
- An abstract implementing class doesn't have to implement the methods but the first concrete subclass must
- A class can only extend one class
- Interfaces and classes can implement multiple interfaces
- Interfaces can only extend interfaces not classes and they cannot implement interfaces
- EXAM TIP: Verify class and interface declarations are legal before looking at other code logic



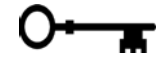
Inheritance

- Inheritance allows a class to be a subclass of a superclass and thereby inherit `public` and `protected` variables and methods of the superclass.
- Inheritance is a key concept that underlies IS-A, polymorphism, overriding, overloading, and casting.
- All classes (except class `Object`) are subclasses of type `Object`, and therefore they inherit `Object`'s methods.



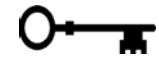
Polymorphism

- Polymorphism means "many forms."
- A reference variable is always of a single, unchangeable type, but it can refer to a subtype object.
- A single object can be referred to by reference variables of many different types—as long as they are the same type or a supertype of the object.
- The reference variable's type (not the object's type) determines which methods can be called!
- Polymorphic method invocations apply only to overridden *instance* methods.



Reference Variable Casting

- There are two types of reference variable casting: downcasting and upcasting.
  - **Downcasting** If you have a reference variable that refers to a subtype object, you can assign it to a reference variable of the subtype. You must make an explicit cast to do this, and the result is that you can access the subtype's members with this new reference variable.
  - **Upcasting** You can assign a reference variable to a supertype reference variable explicitly or implicitly. This is an inherently safe operation because the assignment restricts the access capabilities of the new variable.



Implementing an Interface

- When you implement an interface, you are fulfilling its contract.
- You implement an interface by properly and concretely implementing all of the methods defined by the interface.
- A single class can implement many interfaces.

Self Test (1)

What is the output of the following code?

```
1: abstract class Reptile {
2:   public final void layEggs() { System.out.println("Reptile laying eggs");
   }
3:     public static void main(String[] args) {
4:     Reptile reptile = new Lizard();
5:     reptile.layEggs();
6:     }
7: }
8: public class Lizard extends Reptile {
9:   public void layEggs() { System.out.println("Lizard laying eggs"); }
10: }
```

A. `Reptile laying eggs`
B. `Lizard laying eggs`
C. The code will not compile because of line 4.
D. The code will not compile because of line 5.
E. The code will not compile because of line 9.

Self Test 

Self Test (2)

What is the output of the following code? (Choose all that apply)

```
1: interface Aquatic {
2:   public default int getNumberOfGills(int input) { return 2; }
3: }
4: public class ClownFish implements Aquatic {
5:   public String getNumberOfGills() { return "4"; }
6:   public String getNumberOfGills(int input) { return "6"; }
7:   public static void main(String[] args) {
8:     System.out.println(new ClownFish().getNumberOfGills(-1));
9:   }
10: }
```

A. 2

B. 4

C. 6

D. The code will not compile because of line 5.

E. The code will not compile because of line 6.

F. The code will not compile because of line 8.

Self Test 

Self Test (3)

Which of the following statements about polymorphism are true? (Choose all that apply)

A. A reference to an object may be cast to a subclass of the object without an explicit cast.

B. If a method takes a superclass of three objects, then any of those classes may be passed as a parameter to the method.

C. A method that takes a parameter with type `java.lang.Object` will take any reference.

D. All cast exceptions can be detected at compile-time.

E. By defining a public instance method in the superclass, you guarantee that the specific method will be called in the parent class at runtime.

Self Test 

Self Test (4)

Given:

```
class Rocket {
  private void blastOff() { System.out.print("bang "); }
}
public class Shuttle extends Rocket {
  public static void main(String[] args) {
    new Shuttle().go();
  }
  void go() {
    blastOff();
    // Rocket.blastOff();  // line A
  }
  private void blastOff() { System.out.print("sh-bang "); }
}
```

Which are true? (Choose all that apply.)

A. As the code stands, the output is `bang`
B. As the code stands, the output is `sh-bang`
C. As the code stands, compilation fails.
D. If line A is uncommented, the output is `bang bang`
E. If line A is uncommented, the output is `sh-bang bang`
F. If line A is uncommented, compilation fails.

Self Test 

Self Test (5)

What modifiers are assumed for all interface variables? (Choose all that apply)

A. `public`
B. `protected`
C. `private`
D. `static`
E. `final`
F. `abstract`

Self Test 

Self Test (6)

Given:

```
1. enum Animals {
2.   DOG("woof"), CAT("meow"), FISH("burble");
3.   String sound;
4.   Animals(String s) { sound = s; }
5. }
6. class TestEnum {
7.   static Animals a;
8.   public static void main(String[] args) {
9.     System.out.println(a.DOG.sound + " " + a.FISH.sound);
10.  }
11. }
```

What is the result?

A. `woof burble`
B. Multiple compilation errors
C. Compilation fails due to an error on line 2
D. Compilation fails due to an error on line 3
E. Compilation fails due to an error on line 4
F. Compilation fails due to an error on line 9

Self Test 

Self Test (7)

What is the output of the following code?

```
1: public abstract class Whale {
2:   public abstract void dive() {};
3:   public static void main(String[] args) {
4:     Whale whale = new Orca();
5:     whale.dive();
6:   }
7: }
8: class Orca extends Whale {
9:   public void dive(int depth) { System.out.println("Orca diving"); }
10: }
```

A. `Orca diving`
B. The code will not compile because of line 2.
C. The code will not compile because of line 8.
D. The code will not compile because of line 9.
E. The output cannot be determined from the code provided.

Self Test

Self Test (8)

Which are true? (Choose all that apply.)

A. "X extends Y" is correct if and only if X is a class and Y is an interface.
B. "X extends Y" is correct if and only if X is an interface and Y is a class.
C. "X extends Y" is correct if X and Y are either both classes or both interfaces.
D. "X extends Y" is correct for all combinations of X and Y being classes and/or interfaces.

Self Test

Self Test (9)

Given:

```
1. public class Electronic implements Device
     { public void doIt() { } }
2.
3. abstract class Phone1 extends Electronic { }
4.
5. abstract class Phone2 extends Electronic
     { public void doIt(int x) { } }
6.
7. class Phone3 extends Electronic implements Device
     { public void doStuff() { } }
8.
9. interface Device { public void doIt(); }
```

What is the result? (Choose all that apply.)

A. Compilation succeeds
B. Compilation fails with an error on line 1
C. Compilation fails with an error on line 3
D. Compilation fails with an error on line 5
E. Compilation fails with an error on line 7
F. Compilation fails with an error on line 9

Self Test 

Self Test (10)

Given:

```
4. public class Frodo extends Hobbit
5.   public static void main(String[] args) {
6.     int myGold = 7;
7.     System.out.println(countGold(myGold, 6));
8.   }
9. }
10. class Hobbit {
11.   int countGold(int x, int y) { return x + y; }
12. }
```

What is the result?

A. `13`
B. Compilation fails due to multiple errors
C. Compilation fails due to an error on line 6
D. Compilation fails due to an error on line 7
E. Compilation fails due to an error on line 11

Self Test 

Self Test (11)

Given:

```
interface Gadget {
  void doStuff();
}
abstract class Electronic {
  void getPower() { System.out.print("plug in "); }
}
public class Tablet extends Electronic implements Gadget {
  void doStuff() { System.out.print("show book "); }
  public static void main(String[] args) {
    new Tablet().getPower();
    new Tablet().doStuff();
  }
}
```

Which are true? (Choose all that apply.)

A. The class `Tablet` will NOT compile
B. The interface `Gadget` will NOT compile
C. The output will be `plug in show book`
D. The `abstract` class `Electronic` will NOT compile
E. The class `Tablet` CANNOT both extend and implement

Self Test 

Self Test (12)

What modifiers are implicitly applied to all interface methods? (Choose all that apply)

A. `protected`
B. `public`
C. `static`
D. `void`
E. `abstract`
F. `default`

Self Test 

Self Test (13)

What is the output of the following code?

```
1: class Mammal {
2:   public Mammal(int age) {
3:     System.out.print("Mammal");
4:   }
5: }
6: public class Platypus extends Mammal {
7:   public Platypus() {
8:     System.out.print("Platypus");
9:   }
10:   public static void main(String[] args) {
11:     new Mammal(5);
12:   }
13: }
```

A. `Platypus`
B. `Mammal`
C. `PlatypusMammal`
D. `MammalPlatypus`
E. The code will not compile because of line 8.
F. The code will not compile because of line 11.

Self Test 

Self Test (14)

Which of the following statements can be inserted in the blank line so that the code will compile successfully? (Choose all that apply)

```
public interface CanHop {}
public class Frog implements CanHop {
   public static void main(String[] args) {
     ___________ frog = new TurtleFrog();
   }
}
public class BrazilianHornedFrog extends Frog {}
public class TurtleFrog extends Frog {}
```

A. `Frog`

B. `TurtleFrog`

C. `BrazilianHornedFrog`

D. `CanHop`

E. `Object`

F. `Long`

Self Test 

Self Test (15)

Which statement(s) are correct about the following code? (Choose all that apply)

```
public class Rodent {
  protected static Integer chew() throws Exception {
    System.out.println("Rodent is chewing");
    return 1;
  }
}
public class Beaver extends Rodent {
  public Number chew() throws RuntimeException {
    System.out.println("Beaver is chewing on wood");
    return 2;
  }
}
```

A. It will compile without issue.

B. It fails to compile because the type of the exception the method throws is a subclass of the type of exception the parent method throws.

C. It fails to compile because the return types are not covariant.

D. It fails to compile because the method is `protected` in the parent class and `public` in the subclass.

E. It fails to compile because of a `static` modifier mismatch between the two methods.

Self Test 

Self Test (16)

Which of the following may only be hidden and not overridden? (Choose all that apply)

A. `private` instance methods
B. `protected` instance methods
C. `public` instance methods
D. `static` methods
E. `public` variables
F. `private` variables

Self Test 

Self Test (17)

Choose the correct statement about the following code:

```
1: interface HasExoskeleton {
2:   abstract int getNumberOfSections();
3: }
4: abstract class Insect implements HasExoskeleton {
5:   abstract int getNumberOfLegs();
6: }
7: public class Beetle extends Insect {
8:   int getNumberOfLegs() { return 6; }
9: }
```

A. It compiles and runs without issue.
B. The code will not compile because of line 2.
C. The code will not compile because of line 4.
D. The code will not compile because of line 7.
E. It compiles but throws an exception at runtime.

Self Test 

Self Test (18)

Choose the correct statement about the following code:

```
1: public interface Herbivore {
2:   int amount = 10;
3:   public static void eatGrass();
4:   public int chew() {
5:     return 13;
6:   }
7: }
```

A. It compiles and runs without issue.

B. The code will not compile because of line 2.

C. The code will not compile because of line 3.

D. The code will not compile because of line 4.

E. The code will not compile because of lines 2 and 3.

F. The code will not compile because of lines 3 and 4.

Self Test 

Self Test (19)

Choose the correct statement about the following code:

```
1: public interface CanFly {
2:   void fly();
3: }
4: interface HasWings {
5:   public abstract Object getWindSpan();
6: }
7: abstract class Falcon implements CanFly, HasWings {
8: }
```

A. It compiles without issue.

B. The code will not compile because of line 2.

C. The code will not compile because of line 4.

D. The code will not compile because of line 5.

E. The code will not compile because of lines 2 and 5.

F. The code will not compile because the class `Falcon` doesn't implement the interface methods.

Self Test (20)

Which statements are true for both abstract classes and interfaces? (Choose all that apply)

A. All methods within them are assumed to be abstract.

B. Both can contain `public static final` variables.

C. Both can be extended using the `extend` keyword.

D. Both can contain default methods.

E. Both can contain static methods.

F. Neither can be instantiated directly.

G. Both inherit `java.lang.Object`.

Self Test (21)

What is the output of the following code?

```
1: interface Nocturnal {
2:   default boolean isBlind() { return true; }
3: }
4: public class Owl implements Nocturnal {
5:   public boolean isBlind() { return false; }
6:     public static void main(String[] args) {
7:     Nocturnal nocturnal = (Nocturnal)new Owl();
8:     System.out.println(nocturnal.isBlind());
9:     }
10: }
```

A. `true`

B. `false`

C. The code will not compile because of line 2.

D. The code will not compile because of line 5.

E. The code will not compile because of line 7.

F. The code will not compile because of line 8.

Self Test 

Self Test (22)

What is the output of the following code?

```
1: class Arthropod
2:   public void printName(double input) { System.out
  .print("Arthropod"); }
3: }
4: public class Spider extends Arthropod {
5:   public void printName(int input) { System.out.print("Spider"); }
6:   public static void main(String[] args) {
7:     Spider spider = new Spider();
8:     spider.printName(4);
9:     spider.printName(9.0);
10:   }
11: }
```

A. `SpiderArthropod`

B. `ArthropodSpider`

C. `SpiderSpider`

D. `ArthropodArthropod`

E. The code will not compile because of line 5.

F. The code will not compile because of line 9.

Self Test 

Self Test (23)

Which statements are true about the following code? (Choose all that apply)

```
1: interface HasVocalCords {
2:   public abstract void makeSound();
3: }
4: public interface CanBark extends HasVocalCords {
5:   public void bark();
6: }
```

A. The `CanBark` interface doesn't compile.

B. A class that implements `HasVocalCords` must override the `makeSound()` method.

C. A class that implements `CanBark` inherits both the `makeSound()` and `bark()` methods.

D. A class that implements `CanBark` only inherits the `bark()` method.

E. An interface cannot extend another interface.

Self Test 

Self Test (24)

Which of the following is true about a concrete subclass? (Choose all that apply)

A. A concrete subclass can be declared as `abstract`.

B. A concrete subclass must implement all inherited abstract methods.

C. A concrete subclass must implement all methods defined in an inherited interface.

D. A concrete subclass cannot be marked as `final`.

E. Abstract methods cannot be overridden by a concrete subclass.

Self Test 

Self Test (25)

Which of the following statements can be inserted in the blank so that the code will compile successfully? (Choose all that apply)

```
public class Snake {}
public class Cobra extends Snake {}
public class GardenSnake {}
public class SnakeHandler {
   private Snake snake;
   public void setSnake(Snake snake) { this.snake = snake; }
   public static void main(String[] args) {
     new SnakeHandler().setSnake(__________);
   }
}
```

A. `new Cobra()`

B. `new GardenSnake()`

C. `new Snake()`

D. `new Object()`

E. `new String("Snake")`

F. `null`

Self Test (26)

What is the result of the following code?

```
1: public abstract class Bird {
2:   private void fly() { System.out.println("Bird is flying"); }
3:   public static void main(String[] args) {
4:     Bird bird = new Pelican();
5:     bird.fly();
6:   }
7: }
8: class Pelican extends Bird {
9:   protected void fly() { System.out.println("Pelican is flying"); }
10: }
```

A. `Bird is flying`

B. `Pelican is flying`

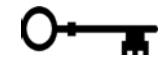
C. The code will not compile because of line 4.

D. The code will not compile because of line 5.

E. The code will not compile because of line 9.

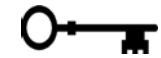
OCA 8 Exam Prep

Section 6:
Exceptions



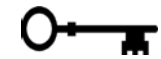
Exceptions (1)

- Exceptions come in two flavors: checked and unchecked.
- Checked exceptions include all subtypes of `Exception`, excluding classes that extend `RuntimeException`.
- Checked exceptions are subject to the handle or declare rule; any method that might throw a checked exception (including methods that invoke methods that can throw a checked exception) must either declare the exception using `throws`, or handle the exception with an appropriate `try/catch`.
- Subtypes of `Error` or `RuntimeException` are unchecked, so the compiler doesn't enforce the handle or declare rule. You're free to handle them or to declare them, but the compiler doesn't care one way or the other.
- If you use an optional `finally` block, it will always be invoked, regardless of whether an exception in the corresponding `try` is thrown or not, and regardless of whether a thrown exception is caught or not.



Exceptions (2)

- The only exception to the `finally`-will-always-be-called rule is that a `finally` will not be invoked if the JVM shuts down. That could happen if code from the `try` or `catch` blocks calls `System.exit()`.
- Just because `finally` is invoked does not mean it will complete. Code in the `finally` block could itself raise an exception or issue a `System.exit()`.
- Uncaught exceptions propagate back through the call stack, starting from the method where the exception is thrown and ending with either the first method that has a corresponding catch for that exception type or a JVM shutdown (which happens if the exception gets to `main()`, and `main()` is "ducking" the exception by declaring it).



Exceptions (3)

- You can create your own exceptions, normally by extending `Exception` or one of its subtypes. Your exception will then be considered a checked exception (unless you are extending from `RuntimeException`), and the compiler will enforce the handle or declare rule for that exception.
- All `catch` blocks must be ordered from most specific to most general. If you have a `catch` clause for both `IOException` and `Exception`, you must put the `catch` for `IOException` first in your code. Otherwise, the `IOException` would be caught by `catch(Exception e)`, because a `catch` argument can catch the specified exception or any of its subtypes! The compiler will stop you from defining `catch` clauses that can never be reached.
- Some exceptions are created by programmers, and some by the JVM.

Key Points

Exceptions (4)

- Common runtime (unchecked) exceptions include:
  - ArithmeticException (JVM)
  - ArrayIndexOutOfBoundsException (JVM)
  - ClassCastException (JVM)
  - IllegalArgumentException (programmer)
  - NullPointerException (JVM)
  - NumberFormatException (programmer)
- Common checked exceptions include:
  - IOException
  - FileNotFoundException
- Common errors include:
  - ExceptionInInitializerError
  - StackOverflowError
  - NoClassDefFoundError

Self Test

Self Test (1)

1. (Also an Upgrade topic) Given:

```
public class Flipper {
  public static void main(String[] args) {
    String o = "-";
    switch("FRED".toLowerCase().substring(1,3)) {
      case "yellow":
        o += "y";
      case "red":
        o += "r";
      case "green":
        o += "g";
    }
    System.out.println(o);
  }
}
```

What is the result?

A. `-`
B. `-r`
C. `-rg`
D. Compilation fails
E. An exception is thrown at runtime

Self Test (2)

2. Given:

```
class Plane {
  static String s = "-";
  public static void main(String[] args) {
    new Plane().s1();
    System.out.println(s);
  }
  void s1() {
    try { s2(); }
    catch (Exception e) { s += "c"; }
  }
  void s2() throws Exception  {
    s3();  s += "2";
    s3();  s += "2b";
  }
  void s3() throws Exception {
    throw new Exception();
  }
}
```

What is the result?

A. `-`
B. `-c`
C. `-c2`
D. `-2c`
E. `-c22b`
F. `-2c2b`
G. `-2c2bc`
H. Compilation fails

Self Test (3)

3. Given:

```
try { int x = Integer.parseInt("two"); }
```

Which could be used to create an appropriate `catch` block? (Choose all that apply.)

A. `ClassCastException`
B. `IllegalStateException`
C. `NumberFormatException`
D. `IllegalArgumentException`
E. `ExceptionInInitializerError`
F. `ArrayIndexOutOfBoundsException`

Self Test (4)

4. Given:

```
public class Flip2 {
  public static void main(String[] args) {
    String o = "-";
    String[] sa = new String[4];
    for(int i = 0; i < args.length; i++)
      sa[i] = args[i];
    for(String n: sa) {
      switch(n.toLowerCase()) {
        case "yellow": o += "y";
        case "red":    o += "r";
        case "green":  o += "g";
      }
    }
    System.out.print(o);
  }
}
```

And given the command-line invocation:

```
Java Flip2 RED Green YeLLow
```

Which are true? (Choose all that apply.)

A. The string `rgy` will appear somewhere in the output
B. The string `rgg` will appear somewhere in the output
C. The string `gyr` will appear somewhere in the output
D. Compilation fails
E. An exception is thrown at runtime

Self Test (5)

5. Given:

```
1. class Loopy {
2.   public static void main(String[] args) {
3.     int[] x = {7,6,5,4,3,2,1};
4.     // insert code here
5.       System.out.print(y + " ");
6.     }
7.   }
8. }
```

Which, inserted independently at line 4, compiles? (Choose all that apply.)

A. `for(int y : x) {`
B. `for(x : int y) {`
C. `int y = 0; for(y : x) {`
D. `for(int y=0, z=0; z<x.length; z++) { y = x[z];`
E. `for(int y=0, int z=0; z<x.length; z++) { y = x[z];`
F. `int y = 0; for(int z=0; z<x.length; z++) { y = x[z];`

Self Test

Self Test (6)

6. Given:

```
class Emu {
  static String s = "-";
  public static void main(String[] args) {
    try {
      throw new Exception();
    } catch (Exception e) {
        try {
          try { throw new Exception();
          } catch (Exception ex) { s += "ic "; }
          throw new Exception(); }
        catch (Exception x) { s += "mc "; }
        finally { s += "mf "; }
    } finally { s += "of "; }
    System.out.println(s);
} }
```

What is the result?

A. `-ic of`
B. `-mf of`
C. `-mc mf`
D. `-ic mf of`
E. `-ic mc mf of`
F. `-ic mc of mf`
G. Compilation fails

Self Test

Self Test (7)

7. Given:

```
3. class SubException extends Exception { }
4. class SubSubException extends SubException { }
5.
6. public class CC { void doStuff() throws SubException { } }
7.
8. class CC2 extends CC { void doStuff() throws SubSubException { } }
9.
10. class CC3 extends CC { void doStuff() throws Exception { } }
11.
12. class CC4 extends CC { void doStuff(int x) throws Exception { } }
13.
14. class CC5 extends CC {  void doStuff()  { } }
```

What is the result? (Choose all that apply.)

A. Compilation succeeds
B. Compilation fails due to an error on line 8
C. Compilation fails due to an error on line 10
D. Compilation fails due to an error on line 12
E. Compilation fails due to an error on line 14

Self Test

Self Test (8)

```
 3. public class Ebb {
 4.   static int x = 7;
 5.   public static void main(String[] args) {
 6.     String s = "";
 7.     for(int y = 0; y < 3; y++) {
 8.       x++;
 9.       switch(x) {
10.         case 8: s += "8 ";
11.         case 9: s += "9 ";
12.         case 10: { s+= "10 "; break; }
13.         default: s += "d ";
14.         case 13: s+= "13 ";
15.       }
16.     }
17.     System.out.println(s);
18.   }
19.   static { x++; }
20. }
```

What is the result?

A. `9 10 d`

B. `8 9 10 d`

C. `9 10 10 d`

D. `9 10 10 d 13`

E. `8 9 10 10 d 13`

F. `8 9 10 9 10 10 d 13`

G. Compilation fails

Self Test

Self Test (9)

9. Given:

```
 3. class Infinity { }
 4. public class Beyond extends Infinity {
 5.   static Integer i;
 6.   public static void main(String[] args) {
 7.     int sw = (int)(Math.random() * 3);
 8.     switch(sw) {
 9.       case 0: {  for(int x = 10; x > 5; x++)
10.                   if(x > 10000000) x = 10;
11.                 break; }
12.       case 1: {  int y = 7 * i;  break;  }
13.       case 2: {  Infinity inf = new Beyond();
14.                  Beyond b = (Beyond)inf;  }
15.     }
16.   }
17. }
```

And given that line 7 will assign the value 0, 1, or 2 to `sw`, which are true? (Choose all that apply.)

A. Compilation fails

B. A `ClassCastException` might be thrown

C. A `StackOverflowError` might be thrown

D. A `NullPointerException` might be thrown

E. An `IllegalStateException` might be thrown

F. The program might hang without ever completing

G. The program will always complete without exception

Self Test (10)

10. Given:

```
3. public class Circles {
4.   public static void main(String[] args) {
5.     int[] ia = {1,3,5,7,9};
6.     for(int x : ia) {
7.       for(int j = 0; j < 3; j++) {
8.         if(x > 4 && x < 8) continue;
9.         System.out.print(" " + x);
10.        if(j == 1) break;
11.        continue;
12.      }
13.      continue;
14.    }
15.  }
16. }
```

What is the result?

A. `1 3 9`

B. `5 5 7 7`

C. `1 3 3 9 9`

D. `1 1 3 3 9 9`

E. `1 1 1 3 3 3 9 9 9`

F. Compilation fails

Self Test (11)

11. Given:

```
3. public class OverAndOver {
4.   static String s = "";
5.   public static void main(String[] args) {
6.     try {
7.       s += "1";
8.       throw new Exception();
9.     } catch (Exception e) { s += "2";
10.    } finally { s += "3"; doStuff(); s += "4";
11.    }
12.    System.out.println(s);
13.  }
14.  static void doStuff() { int x = 0; int y = 7/x; }
15. }
```

What is the result?

A. `12`

B. `13`

C. `123`

D. `1234`

E. Compilation fails

F. `123` followed by an exception

G. `1234` followed by an exception

H. An exception is thrown with no other output

Self Test (12)

12. Given:

```
 3. public class Wind {
 4.   public static void main(String[] args) {
 5.     foreach:
 6.     for(int j=0; j<5; j++) {
 7.       for(int k=0; k< 3; k++) {
 8.         System.out.print(" " + j);
 9.         if(j==3 && k==1) break foreach;
10.         if(j==0 || j==2) break;
11.       }
12.     }
13.   }
14. }
```

What is the result?

A. `0 1 2 3`

B. `1 1 1 3 3`

C. `0 1 1 1 2 3 3`

D. `1 1 1 3 3 4 4 4`

E. `0 1 1 1 2 3 3 4 4 4`

F. Compilation fails

Self Test (13)

13. Given:

```
 3. public class Gotcha {
 4.   public static void main(String[] args) {
 5.     // insert code here
 6.
 7.   }
 8.   void go() {
 9.     go();
10.   }
11. }
```

And given the following three code fragments:

```
I.   new Gotcha().go();

II.  try { new Gotcha().go(); }
     catch (Error e) { System.out.println("ouch"); }

III. try { new Gotcha().go(); }
     catch (Exception e) { System.out.println("ouch"); }
```

When fragments I–III are added, independently, at line 5, which are true? (Choose all that apply.)

A. Some will not compile

B. They will all compile

C. All will complete normally

D. None will complete normally

E. Only one will complete normally

F. Two of them will complete normally

Self Test  Self Test (14)

14. Given the code snippet:

```
String s = "bob";
String[] sa = {"a", "bob"};
final String s2 = "bob";
StringBuilder sb = new StringBuilder("bob");

// switch(sa[1]) {          // line 1
// switch("b" + "ob") {     // line 2
// switch(sb.toString()) {  // line 3

// case "ann":  ;           // line 4
// case s:      ;           // line 5
// case s2:     ;           // line 6
}
```

And given that the numbered lines will all be tested by un-commenting one `switch` statement and one `case` statement together, which line(s) will FAIL to compile? (Choose all that apply.)

A. line 1
B. line 2
C. line 3
D. line 4
E. line 5
F. line 6
G. All six lines of code will compile

Self Test  Self Test (15)

15. Given:

```
1. public class Frisbee {
2.   // insert code here
3.     int x = 0;
4.     System.out.println(7/x);
5.   }
6. }
```

And given the following four code fragments:

```
I.   public static void main(String[] args) {
II.  public static void main(String[] args) throws Exception {
III. public static void main(String[] args) throws IOException {
IV.  public static void main(String[] args) throws RuntimeException {
```

If the four fragments are inserted independently at line 2, which are true? (Choose all that apply.)

A. All four will compile and execute without exception
B. All four will compile and execute and throw an exception
C. Some, but not all, will compile and execute without exception
D. Some, but not all, will compile and execute and throw an exception
E. When considering fragments II, III, and IV, of those that will compile, adding a `try/catch` block around line 4 will cause compilation to fail

Self Test (16)

16. Given:

```
2. class MyException extends Exception { }
3. class Tire {
4.   void doStuff() {  }
5. }
6. public class Retread extends Tire {
7.   public static void main(String[] args) {
8.     new Retread().doStuff();
9.   }
10.  // insert code here
11.    System.out.println(7/0);
12.  }
13. }
```

And given the following four code fragments:

```
I.   void doStuff() {
II.  void doStuff() throws MyException {
III. void doStuff() throws RuntimeException {
IV.  void doStuff() throws ArithmeticException {
```

When fragments I–IV are added, independently, at line 10, which are true? (Choose all that apply.)

A. None will compile
B. They will all compile
C. Some, but not all, will compile
D. All of those that compile will throw an exception at runtime
E. None of those that compile will throw an exception at runtime
F. Only some of those that compile will throw an exception at runtime