**O1**

What is the output for the below code ?

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Test {

public static void main(String... args) {

Pattern p = Pattern.compile("a\*b");

Matcher m = p.matcher("b");

boolean b = m.matches();

System.out.println(b);

}

}

options

A)true

B)Compile Error

C)false

D)b

Correct answer is : A

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | **O2**  What is the output for the below code ?  public class Test {    public static void main(String... args) {      String input = "1 fish 2 fish red fish blue fish";  Scanner s = new Scanner(input).  useDelimiter("\\s\*fish\\s\*");  System.out.println(s.nextInt());  System.out.println(s.nextInt());  System.out.println(s.next());  System.out.println(s.next());  s.close();    }  }  options  A)1 2 red blue  B)Compile Error - because Scanner is not defind in java.  C)1 fish 2 fish red fish blue fish  D)1 fish 2 fish red blue fish  Correct answer is : A  Explanations : java.util.Scanner is a simple text scanner which can parse primitive types and strings using regular expressions.  **O3**  What is the output for the below code ?  public class Test {    public static void main(String... args) {    Pattern p = Pattern.compile("a{3}b?c\*");  Matcher m = p.matcher("aaab");  boolean b = m.matches();  System.out.println(b);  }  }  options  A)true  B)Compile Error  C)false  D)NullPointerException  Correct answer is : A  Explanations :  X? X, once or not at all  X\* X, zero or more times  X+ X, one or more times  X{n} X, exactly n times  X{n,} X, at least n times  X{n,m} X, at least n but not more than m times  **O4**  What is the output for the below code ?  public class Test {    public static void main(String... args) {    Pattern p = Pattern.compile("a{1,3}b?c\*");  Matcher m = p.matcher("aaab");  boolean b = m.matches();  System.out.println(b);    }  }  options  A)true  B)Compile Error  C)false  D)NullPointerException  Correct answer is : A  Explanations :  X? X, once or not at all  X\* X, zero or more times  X+ X, one or more times  X{n} X, exactly n times  X{n,} X, at least n times  X{n,m} X, at least n but not more than m times |  | |  |

**O5**

Given:

11. String test= “a1b2c3”;

12. String[] tokens = test.split(”\\d”);

13. for(String s: tokens) System.out.print(s +“ “);

What is the result?

A. a b c

B. 1 2 3

C. a1b2c3

D. a1 b2 c3

E. Compilation fails.

F. The code runs with no output.

G. An exception is thrown at runtime.

Answer: A

**O6**

Given:

11. String test = “Test A. Test B. Test C.”;

12. // insert code here

13. String[] result = test.split(regex);

Which regular expression inserted at line 12 will correctly split test into

“Test A”, “Test B”, and “Test C”?

A. String regex = “”;

B. String regex = “ “;

C. String regex = “.\*“.

D. String regex = “\\s”

E. String regex = “\\.\\s\*”;

F. String regex = “\\w[ \.] +“;

Answer: E

**O7**

Given:

11. String test = “This is a test”;

12. String[] tokens = test.split(”\s”);

What is the result?

A. 0

B. 1

C. 4

D. Compilation fails.

E. An exception is thrown at runtime.

Answer: D

**O8**

What is the output for the below code ?

public class Outer {

private int a = 7;

class Inner {

public void displayValue() {

System.out.println("Value of a is " + a);

}

}

}

public class Test {

public static void main(String... args) throws Exception {

Outer mo = new Outer();

Outer.Inner inner = mo.new Inner();

inner.displayValue();

}

}

options

A)Value of a is 7

B)Compile Error - not able to access private member.

C)Runtime Exception

D)Value of a is 8

Correct answer is : A

Explanations : An inner class instance can never stand alone without a direct relationship to an instance of the outer class.

you can access the inner class is through a live instance of the outer class.

Inner class can access private member of the outer class.

**O9**

What is the output for the below code ?

public class Tech {

public void tech() {

System.out.println("Tech");

}

}

public class Atech {

Tech a = new Tech() {

public void tech() {

System.out.println("anonymous tech");

}

};

public void dothis() {

a.tech();

}

public static void main(String... args){

Atech atech = new Atech();

atech.dothis();

}

options

A)anonymous tech

B)Compile Error

C)Tech

D)anonymous tech Tech

Correct answer is : A

Explanations : This is anonymous subclass of the specified class type.

Anonymous inner class ( anonymous subclass ) overriden the Tech super class of tech() method.

Therefore Subclass method will get called.

**10**

What is the output for the below code ?

public class Outer {

private String x = "Outer variable";

void doStuff() {

String z = "local variable";

class Inner {

public void seeOuter() {

System.out.println("Outer x is " + x);

System.out.println("Local variable z is " + z);

}

}

}

}

options

A)Outer x is Outer variable.

B)Compile Error

C)Local variable z is local variable.

D)Outer x is Outer variable Local variable z is local variable

Correct answer is : B

Explanations : Cannot refer to a non-final variable z inside an inner class defined in a different method.

**11**

46. Given:

10. class Line {

11. public static class Point { }

12. }

13.

14. class Triangle {

15. // insert code here

16. }

Which code, inserted at line 15, creates an instance of the Point class

defined in Line?

A. Point p = new Point();

B. Line.Point p = new Line.Point();

C. The Point class cannot be instatiated at line 15.

D. Line 1 = new Line() ; 1.Point p = new 1.Point();

Answer: B

**12**

Given:

11. static classA {

12. void process() throws Exception { throw new Exception(); }

13. }

14. static class B extends A {

15. void process() { System.out.println(”B “); }

16. }

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

18.A a=new B();

19. a.process();

20.}

What is the result?

A. B

B. The code runs with no output.

C. An exception is thrown at runtime.

D. Compilation fails because of an error in line 15.

E. Compilation fails because of an error in line 18.

F. Compilation fails because of an error in line 19.

Answer: F

**13**

Given:

1. package geometry;

2. public class Hypotenuse {

3. public InnerTriangle it = new InnerTriangle();

4. class InnerTriangle {

5. public int base;

6. public int height;

7. }

8. }

Which is true about the class of an object that can reference the

variable base?

A. It can be any class.

B. No class has access to base.

C. The class must belong to the geometry package.

D. The class must be a subclass of the class Hypotenuse.

Answer: C

**14 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1. import java.util.\*;**

**2. public class Old {**

**3. public static Object get0(List list) {**

**4. return list.get(0);**

**5. }**

**6. }**

**Which three will compile successfully? (Choose three.)**

A. Object o = Old.get0(new LinkedList());

B. Object o = Old.get0(new LinkedList<?>());

C. String s = Old.get0(new LinkedList<String>());

D. Object o = Old.get0(new LinkedList<Object>());

E. String s = (String)Old.get0(new LinkedList<String>());

**Answer: A, D, E**

**15\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**11. public static int sum(List list) {**

**12. int sum = 0;**

**13. for ( Iterator iter = list.iterator(); iter.hasNext(); ) {**

**14. int i = ((Integer)iter.next()).intValue();**

**15. sum += i;**

**16. }**

**17. return sum;**

**18. }**

**Which three changes must be made to the method sum to use generics? (Choose three.)**

A. remove line 14

B. replace line 14 with "int i = iter.next();"

C. replace line 13 with "for (int i : intList) {"

D. replace line 13 with "for (Iterator iter : intList) {"

E. replace the method declaration with "sum(List<int> intList)"

F. replace the method declaration with "sum(List<Integer> intList)"

**Answer: A, C, F**

**16\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**11. // insert code here**

**12. private N min, max;**

**13. public N getMin() { return min; }**

**14. public N getMax() { return max; }**

**15. public void add(N added) {**

**16. if (min == null || added.doubleValue() < min.doubleValue()) 17. min = added;**

**18. if (max == null || added.doubleValue() > max.doubleValue()) 19. max = added;**

**20. }**

**21. }**

**Which two, inserted at line 11, will allow the code to compile? (Choose two.)**

A. public class MinMax<?> {

B. public class MinMax<? extends Number> {

C. public class MinMax<N extends Object> {

D. public class MinMax<N extends Number> {

E. public class MinMax<? extends Object> {

F. public class MinMax<N extends Integer> {

**Answer: D, F**

**17\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**10. interface A { void x(); }**

**11. class B implements A { public void x() {} public void y() {} }**

**12. class C extends B { public void x() {} }**

**And:**

**20. java.util.List<A> list = new java.util.ArrayList<A>();**

**21. list.add(new B());**

**22. list.add(new C());**

**23. for (A a : list) {**

**24. a.x();**

**25. a.y();**

**26. }**

**What is the result?**

A. The code runs with no output.

B. An exception is thrown at runtime.

C. Compilation fails because of an error in line 20.

D. Compilation fails because of an error in line 21.

E. Compilation fails because of an error in line 23.

F. Compilation fails because of an error in line 25.

**Answer: F**

**18\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**11. public static void append(List list) { list.add("0042"); }**

**12. public static void main(String[] args) {**

**13. List<Integer> intList = new ArrayList<Integer>();**

**14. append(intList);**

**15. System.out.println(intList.get(0));**

**16. }**

**What is the result?**

A. 42

B. 0042

C. An exception is thrown at runtime.

D. Compilation fails because of an error in line 13.

E. Compilation fails because of an error in line 14.

**Answer: B**

**19\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**A programmer must create a generic class MinMax**

**and the type parameter**

**of MinMax must implement Comparable.**

**Which implementation of MinMax will compile?**

A. class MinMax<E extends Comparable<E>> {

E min = null;

E max = null;

public MinMax() {}

public void put(E value) { /\* store min or max \*/ }

B. class MinMax<E implements Comparable<E>> {

E min = null;

E max = null;

public MinMax() {}

public void put(E value) { /\* store min or max \*/ }

C. class MinMax<E extends Comparable<E>> {

<E> E min = null;

<E> E max = null;

public MinMax() {}

public <E> void put(E value) { /\* store min or max \*/ }

D. class MinMax<E implements Comparable<E>> {

<E> E min = null;

<E> E max = null;

public MinMax() {}

public <E> void put(E value) { /\* store min or max \*/ }

**Answer: A**

**20\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Given:**

**1. public class Score implements Comparable<Score> {**

**2. private int wins, losses;**

**3. public Score(int w, int 1) { wins = w; losses = 1; }**

**4. public int getWins() { return wins; }**

*5.* **public int getLosses() { return losses; }**

**6. public String toString() {**

**7. return “<“ + wins + “,“ + losses + “>”;**

**8. }**

**9. // insert code here**

**10. }**

**Which method will complete this class?**

A. public int compareTo(Object o) {/\*mode code here\*/}

B. public int compareTo(Score other) {/\*more code here\*/}

C. public int compare(Score s1,Score s2){/\*more code here\*/}

D. public int compare(Object o1,Object o2){/\*more code here\*/}

**Answer: B**

**21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Given:**

**1. import java.util.\*;**

**2. public class Test {**

**3. public static void main(String[] args) {**

**4. List<String> strings = new ArrayList<String>();**

*5.* **// insert code here**

**6. }**

**7. }**

**Which four, inserted at line** *5,* **will allow compilation to succeed?**

**(Choose four.)**

A. String s = strings.get(0);

B. Iterator i1 = strings.iterator();

C. String[] array1 = strings.toArray();

D. Iterator<String> i2 = strings.iterator();

E. String[] array2 = strings.toArray(new String[1]);

F. **Iterator<String> i3 = strings.iterator<String>();**

**Answer: ABDE**

**22\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Given:**

**classA {}**

**class B extends A {}**

**class C extends A {}**

**class D extends B {}**

**Which three statements are true? (Choose three.)**

A. The type List<A> is assignable to List.

B. The type List<B> is assignable to List<A>.

C. The type List<Object> is assignable to List<?>.

D. The type List<D> is assignable to List<? extends B>.

E. The type List<? extends A> is assignable to List<A>.

F. The type List<Object> is assignable to any List reference.

G. The type List<? extends B> is assignable to List<? extends A>.

**Answer: CDG**

**23\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Given:**

**11. public void addStrings(List list) {**

**12. list.add(”foo”);**

**13. list.add(”bar”);**

**14. }**

**What must you change in this method to compile without warnings?**

A. add this code after line 11:

list = (List<String>) list;

B. change lines 12 and 13 to:

list.add<String>(”foo”);

list.add<String>(”bar”);

C. change the method signature on line 11 to:

public void addStrings(List<? extends String> list) {

D. change the method signature on line 11 to:

public void addStrings(List<? super String> list) {

E. No changes are necessary. This method compiles without warnings.

**Answer: D**

**24\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Given:**

**1. public class Test {**

**2. public <T extends Comparable> T findLarger(T x, T y) {**

**3. if(x.compareTo(y) > 0) {**

**4. return x;**

*5.* **} else {**

**6. return y;**

**7. }**

**8. }**

**9. }**

**and:**

**22. Test t = new Test();**

**23. // insert code here**

**Which two will compile without errors when inserted at line 23?**

**(Choose two.)**

A. Object x = t.findLarger(123, *“456”);*

B. int x = t.findLarger(123, new Double(456));

C. int x = t.findLarger(123, new Integer(456));

D. int x = (int) t.findLarger(new Double(123), new Double(456));

**Answer: AC**

**25\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**interface Hungry<E> { void munch(E x); }**

**interface Carnivore<E extends Animal> extends Hungry<E> {}**

**interface Herbivore<E extends Plant> extends Hungry<E> {}**

**abstract class Plant {}**

**class Grass extends Plant {}**

**abstract class Animal {}**

**class Sheep extends Animal implements Herbivore<Sheep> {**

**public void munch(Sheep x) {}**

**}**

**class Wolf extends Animal implements Carnivore<Sheep> {**

**public void munch(Sheep x) {}**

**}**

**Which of the following changes (taken separately) would allow this code to compile?**

**(Choose all that apply.)**

A. Change the Carnivore interface to

interface Carnivore<E extends Plant> extends Hungry<E> {}

B. Change the Herbivore interface to

interface Herbivore<E extends Animal> extends Hungry<E> {}

C. Change the Sheep class to

class Sheep extends Animal implements Herbivore<Plant> {

public void munch(Grass x) {}

}

D. Change the Sheep class to

class Sheep extends Plant implements Carnivore<Wolf> {

public void munch(Wolf x) {}

}

E. Change the Wolf class to

class Wolf extends Animal implements Herbivore<Grass> {

public void munch(Grass x) {}

}

F. No changes are necessary.

**Answer:**

-> B is correct. The problem with the original code is that Sheep tries to implement

Herbivore<Sheep> and Herbivore declares that its type parameter E can be any type that extends Plant. Since a Sheep is not a Plant, Herbivore<Sheep> makes no sense— the type Sheep is outside the allowed range of Herbivore's parameter E. Only solutions that either alter the definition of a Sheep or alter the definition of Herbivore will be able to fix this. So A, E, and F are eliminated. B works, changing the definition of an Herbivore to allow it to eat Sheep solves the problem. C doesn't work because an Herbivore<Plant> must have a munch(Plant) method, not munch(Grass).

And D doesn't work, because in D we made Sheep extend Plant, now the Wolf class breaks because its munch(Sheep) method no longer fulfills the contract of Carnivore.

**26**

Given the code. What is the result?

public class TrickyNum<X extends Number> {

private X x;

public TrickyNum(X x) {

this.x = x;

}

private double getDouble() {

return x.doubleValue();

}

public static void main(String args[]) {

TrickyNum<Integer> a = new TrickyNum<Integer>(new Integer(1));

System.out.print(a.getDouble());

}

}

A) Compilation fails.

B) An exception is thrown at runtime.

C) "1.0" is printed.

D) "1" is printed.

Resposta: C

**27**



Resposta:

import java.util.\*;

public class StringToInteger {

HashMap<String, Integer> hm = new HashMap<String, Integer>();

public void add(String a, Integer b) {

hm.put(a, b);

}

public Set<String> getKeys() {

return hm.keySet();

}

public Collection<Integer> getValues() {

return hm.values();

}

public static void main(String args[]) {

StringToInteger si = new StringToInteger();

si.add("one", 1);

si.add("two", 2);

System.out.println(si.getKeys());

System.out.println(si.getValues());

}

}

**28**

Given the code. Select correct calls of methods. (Select all that apply)

import java.util.\*;

class Empty {

}

class Extended extends Empty {

}

public class TryMe {

public static void doStuff1(List<Empty> list) {

// some code

}

public static void doStuff2(List list) {

// some code

}

public static void doStuff3(List<? extends Empty> list) {

// some code

}

public static void main(String args[]) {

List<Empty> list1 = new LinkedList<Empty>();

List<Extended> list2 = new LinkedList<Extended>();

// more code here

}

}

A) doStuff1(list1);

B) doStuff2(list2);

C) doStuff2(list1);

D) doStuff3(list1);

E) doStuff3(list2);

F) doStuff1(list2);

Answers: A, B, C, D, E

**29**

**30**

**31**

**32**

**O1**