Generics

1 – Java Generics

1. What will be the output of the following Java code?

import java.util.\*;

public class genericstack <E>

{

Stack <E> stk = new Stack <E>();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack <String> gs = new genericstack<String>();

gs.push("Hello");

System.out.println(gs.pop());

}

}

a) H

b) Hello

c) Runtime Error

d) Compilation Error

Answer: b

Explanation: None.

2. What will be the output of the following Java code?

import java.util.\*;

public class genericstack <E>

{

Stack <E> stk = new Stack <E>();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack <Integer> gs = new genericstack<Integer>();

gs.push(36);

System.out.println(gs.pop());

}

}

a) 0

b) 36

c) Runtime Error

d) Compilation Error

Answer: b

Explanation: None.

3. What will be the output of the following Java code?

import java.util.\*;

public class genericstack <E>

{

Stack <E> stk = new Stack <E>();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack <String> gs = new genericstack<String>();

gs.push("Hello");

System.out.print(gs.pop() + " ");

genericstack <Integer> gs = new genericstack<Integer>();

gs.push(36);

System.out.println(gs.pop());

}

}

a) Error

b) Hello

c) 36

d) Hello 36

Answer: d

Explanation: None.

4. What will be the output of the following Java program?

import java.util.\*;

public class genericstack <E>

{

Stack <E> stk = new Stack <E>();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack <String> gs = new genericstack<String>();

gs.push(36);

System.out.println(gs.pop());

}

}

a) 36

b) Hello

c) Runtime Error

d) Compilation Error

Answer: d

Explanation: generic stack object gs is defined to contain a string parameter but we are sending an integer parameter, which results in compilation error.

5. Which of these Exception handlers cannot be type parameterized?

a) catch

b) throw

c) throws

d) all of the mentioned

Answer: d

Explanation: we cannot Create, Catch, or Throw Objects of Parameterized Types as generic class cannot extend the Throwable class directly or indirectly.

6. Which of the following cannot be Type parameterized?

a) Overloaded Methods

b) Generic methods

c) Class methods

d) Overriding methods

Answer: a

Explanation: Cannot Overload a Method Where the Formal Parameter Types of Each Overload Erase to the Same Raw Type.

2 – Generic Methods

1. What are generic methods?

a) Generic methods are the methods defined in a generic class

b) Generic methods are the methods that extend generic class methods

c) Generic methods are methods that introduce their own type parameters

d) Generic methods are methods that take void parameters

Answer: c

Explanation: Generic methods are methods that introduce their own type parameters. This is similar to declaring a generic type, but the type parameter scope is limited to the method where it is declared. Static and non-static generic methods are allowed, as well as generic class constructors.

2. Which of these type parameters is used for a generic methods to return and accept any type of object?

a) K

b) N

c) T

d) V

Answer: c

Explanation: T is used for type, A type variable can be any non-primitive type you specify: any class type, any interface type, any array type, or even another type variable.

3. Which of these type parameters is used for a generic methods to return and accept a number?

a) K

b) N

c) T

d) V

Answer: b

Explanation: N is used for Number.

4. Which of these is an correct way of defining generic method?

a) <T1, T2, …, Tn> name(T1, T2, …, Tn) { /\* … \*/ }

b) public <T1, T2, …, Tn> name<T1, T2, …, Tn> { /\* … \*/ }

c) class <T1, T2, …, Tn> name[T1, T2, …, Tn] { /\* … \*/ }

d) <T1, T2, …, Tn> name{T1, T2, …, Tn} { /\* … \*/ }

Answer: b

Explanation: The syntax for a generic method includes a type parameter, inside angle brackets, and appears before the method’s return type. For static generic methods, the type parameter section must appear before the method’s return type.

5. Which of the following allows us to call generic methods as a normal method?

a) Type Interface

b) Interface

c) Inner class

d) All of the mentioned

Answer: a

Explanation: Type inference, allows you to invoke a generic method as an ordinary method, without specifying a type between angle brackets.

6. What will be the output of the following Java program?

import java.util.\*;

public class genericstack <E>

{

Stack <E> stk = new Stack <E>();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack <String> gs = new genericstack<String>();

gs.push("Hello");

System.out.println(gs.pop());

}

}

a) H

b) Hello

c) Runtime Error

d) Compilation Error

Answer: b

Explanation: None.

7. What will be the output of the following Java program?

import java.util.\*;

public class genericstack <E>

{

Stack <E> stk = new Stack <E>();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack <Integer> gs = new genericstack<Integer>();

gs.push(36);

System.out.println(gs.pop());

}

}

a) 0

b) 36

c) Runtime Error

d) Compilation Error

Answer: b

Explanation: None.

8. What will be the output of the following Java program?

import java.util.\*;

public class genericstack <E>

{

Stack <E> stk = new Stack <E>();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack <String> gs = new genericstack<String>();

gs.push("Hello");

System.out.print(gs.pop() + " ");

genericstack <Integer> gs = new genericstack<Integer>();

gs.push(36);

System.out.println(gs.pop());

}

}

a) Error

b) Hello

c) 36

d) Hello 36

Answer: d

Explanation: None.

3 – Restrictions on Generics

1. Which of these types cannot be used to initiate a generic type?

a) Integer class

b) Float class

c) Primitive Types

d) Collections

Answer: c

Explanation: None.

2. Which of these instance cannot be created?

a) Integer instance

b) Generic class instance

c) Generic type instance

d) Collection instances

Answer: c

Explanation: It is not possible to create generic type instances. Example – “E obj = new E()” will give a compilation error.

3. Which of these data type cannot be type parameterized?

a) Array

b) List

c) Map

d) Set

Answer: a

Explanation: None.

4. What will be the output of the following Java program?

public class BoxDemo

{

public static <U> void addBox(U u,

java.util.List<Box<U>> boxes)

{

Box<U> box = new Box<>();

box.set(u);

boxes.add(box);

}

public static <U> void outputBoxes(java.util.List<Box<U>> boxes)

{

int counter = 0;

for (Box<U> box: boxes)

{

U boxContents = box.get();

System.out.println("Box #" + counter + " contains [" + boxContents.toString() + "]");

counter++;

}

}

public static void main(String[] args)

{

java.util.ArrayList<Box<Integer>> listOfIntegerBoxes = new java.util.ArrayList<>();

BoxDemo.<Integer>addBox(Integer.valueOf(10), listOfIntegerBoxes);

BoxDemo.outputBoxes(listOfIntegerBoxes);

}

}

a) 10

b) Box #0 [10]

c) Box contains [10]

d) Box #0 contains [10]

Answer: d

Explanation: None.

5. What will be the output of the following Java program?

import java.util.\*;

public class genericstack <E>

{

Stack <E> stk = new Stack <E>();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack <String> gs = new genericstack<String>();

gs.push("Hello");

System.out.print(gs.pop() + " ");

genericstack <Integer> gs = new genericstack<Integer>();

gs.push(36);

System.out.println(gs.pop());

}

}

a) Error

b) Hello

c) 36

d) Hello 36

Answer: d

Explanation: None.

6. What will be the output of the following Java program?

import java.util.\*;

class Output

{

public static double sumOfList(List<? extends Number> list)

{

double s = 0.0;

for (Number n : list)

s += n.doubleValue();

return s;

}

public static void main(String args[])

{

List<Double> ld = Arrays.asList(1.2, 2.3, 3.5);

System.out.println(sumOfList(ld));

}

}

a) 5.0

b) 7.0

c) 8.0

d) 6.0

Answer: b

Explanation: None.

7. What will be the output of the following Java program?

import java.util.\*;

class Output

{

public static void addNumbers(List<? super Integer> list)

{

for (int i = 1; i <= 10; i++)

{

list.add(i);

}

}

public static void main(String args[])

{

List<Double> ld = Arrays.asList();

addnumbers(10.4);

System.out.println("getList(2)");

}

}

a) 1

b) 2

c) 3

d) 6

Answer: a

Explanation: None.

8. What will be the output of the following Java program?

import java.util.\*;

public class genericstack <E>

{

Stack <E> stk = new Stack <E>();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack <Integer> gs = new genericstack<Integer>();

gs.push(36);

System.out.println(gs.pop());

}

}

a) H

b) Hello

c) Runtime Error

d) Compilation Error

Answer: d

Explanation: generic stack object gs is defined to contain a string parameter but we are sending an integer parameter, which results in compilation error.

4 – Wildcards

1. Which of these is wildcard symbol?

a) ?

b) !

c) %

d) &

Answer: a

Explanation: In generic code, the question mark (?), called the wildcard, represents an unknown type.

2. What is use of wildcards?

a) It is used in cases when type being operated upon is not known

b) It is used to make code more readable

c) It is used to access members of super class

d) It is used for type argument of generic method

Answer: a

Explanation: The wildcard can be used in a variety of situations: as the type of a parameter, field, or local variable; sometimes as a return type (though it is better programming practice to be more specific). The wildcard is never used as a type argument for a generic method invocation, a generic class instance creation, or a supertype.

3. Which of these keywords is used to upper bound a wildcard?

a) stop

b) bound

c) extends

d) implements

Answer: c

Explanation: None.

4. Which of these is an correct way making a list that is upper bounded by class Number?

a) List<? extends Number>

b) List<extends ? Number>

c) List(? extends Number)

d) List(? UpperBounds Number)

Answer: a

Explanation: None.

5. Which of the following keywords are used for lower bounding a wild card?

a) extends

b) super

c) class

d) lower

Answer: b

Explanation: A lower bounded wildcard is expressed using the wildcard character (‘?’), following by the super keyword, followed by its lower bound: .

6. What will be the output of the following Java program?

import java.util.\*;

class Output

{

public static double sumOfList(List<? extends Number> list)

{

double s = 0.0;

for (Number n : list)

s += n.doubleValue();

return s;

}

public static void main(String args[])

{

List<Integer> li = Arrays.asList(1, 2, 3);

System.out.println(sumOfList(li));

}

}

a) 0

b) 4

c) 5.0

d) 6.0

Answer: d

Explanation: None.

7. What will be the output of the following Java program?

import java.util.\*;

class Output

{

public static double sumOfList(List<? extends Number> list)

{

double s = 0.0;

for (Number n : list)

s += n.doubleValue();

return s;

}

public static void main(String args[])

{

List<Double> ld = Arrays.asList(1.2, 2.3, 3.5);

System.out.println(sumOfList(ld));

}

}

a) 5.0

b) 7.0

c) 8.0

d) 6.0

Answer: b

Explanation: None.

8. What will be the output of the following Java program?

import java.util.\*;

public class genericstack <E>

{

Stack <E> stk = new Stack <E>();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack <Integer> gs = new genericstack<Integer>();

gs.push(36);

System.out.println(gs.pop());

}

}

a) H

b) Hello

c) Runtime Error

d) Compilation Error

Answer: d

Explanation: generic stack object gs is defined to contain a string parameter but we are sending an integer parameter, which results in compilation error.