Generic Types and Collections

Object Orientated Programming in Java

Benjamin Kenwright

Outline

- Review
- Essential Generic Type Concepts
- Collections with Java
- Today’s Practical
- Review/Discussion

Question

- Does the following code compile? What would the output be?

```
public class QuestionA
{
    public static void main(String args[] )
    {
        new Question().Go1();
    }
    void Go1()
    {
        int x = 5;
        Go2(++x);
    }
    void Go2(int y)
    {
        int x = ++y;
        System.out.println(x);
    }
}
```

Answer

- 7

```
C:>java -cp . QuestionA
7
```

Question

- Does the following code compile? What would the output be?

```
public class QuestionA
{
    public static void main(String args[] )
    {
        System.out.println("val:" + 1 + 2);
        System.out.println(1 + 2 + ":val");
    }
}
```

Answer

val:12

3:val

```
C:\>java -cp . QuestionA
val:12
3:val
```

Question

- What will the following code print?

```
public class QuestionA
{
    public static void main(String args[] )
    {
        int x = 1;
        for (int i=0; i<3; i++)
        {
            x += 5 * i;
        }
        System.out.println(x);
    }
}
```

a) 1
b) 10
c) 16
d) 31

Answer

- c) 16

Question

- Does the following code compile? What would the output be?

```
public class QuestionA
{
    public static void main(String args[] )
    {
        int x = 1;
        for (int i=0; i<(int)'a'; i++)
        {
            x = i;
        }
        System.out.println(x);
    }
}
```

Answer

- 96

```
C:\>java -cp . QuestionA
96
```

Question

- Does the following code compile? What would the output be?

```
public class QuestionA
{
    public static void main(String args[] )
    {
        int sum = 0;
        int i = 0;
        while (i < 5)
        {
            sum = sum + i;
            i++;
        }
        System.out.print(i);
        System.out.print(" ");
        System.out.print(sum);
    }
}
```

Answer

- 5 10

Question

- Does the following code compile? What would the output be?

a) (that is, the empty string, printed twice)
b) *
c) !***
d) !****
e) !!!***

```
public class QuestionA
{
    public static void main(String args[] )
    {
        String S = ""; String T = "";
        int i = 4;
        for (i = 1; i <= 3; i++);
        S = S + "!";
        for (i = 1; i < 4; i++)
        T = T + "*";
        System.out.print(S);
        System.out.println(T);
    }
}
```

Answer

- c) !***

Question

- What is the output of the following program?

```
public class QuestionA
{
    public static void main(String[] args)
    {
        double X = 123.321;
        String Y = "Hi!";
        System.out.format("%7.3f%s", X, Y);
    }
}
```

A. `23.32Hi!`
B. `123.321Hi!`
C. `+23.32Hi!`
D. `1.23e2Hi!`
E. none; a compile- or run-time error occurs

Answer

- B) 123.321Hi!

Generic Types

- Generics is the capability to parameterize types
- Flexibility to define a class or a method with generic types that the compiler can replace with concrete types

Example

```
package java.lang;

public interface Comparable {
  public int compareTo(Object o)
}
```

```
package java.lang;

public interface Comparable<T> {
  public int compareTo(T o)
}
```

- <T> represents the formal generic type
- Replaced by an actual concrete type

Why use Generic Types?

Why use Generics?

- Identify errors at compile time
- Explicit type checking
- Robust and reliable programs

Example

- Arrays

  The `ArrayList` Class

- Create array for `strings':

```
ArrayList<String> list = new ArrayList<String>();
```

- Only add `strings' to the array

Example

```
import java.util.ArrayList;
public class QuestionA
{
    public static void main(String args[] )
    {
        ArrayList<String> list = new ArrayList<String>();
        list.add("2");
        list.add("cat");

        for (int i=0; i<list.size(); i++)
        {
            System.out.println( list.get(i) );
        }// End for i
    }// End void main(..)
}// End class
```

Writing Generic Class

- Specify the ‘Type’ in the class definition
  - e.g., <E>, <T>, …
- Use the Type as needed

Example

```
class GenericStack<E>
{
    private ArrayList<E> list = new ArrayList<E>();
    public int getSize()
    {
        return list.size();
    }
    public void push(E o)
    {
        list.add(o);
    }
    public E pop()
    {
        E o = list.get(getSize()-1);
        list.remove(getSize()-1);
        return o;
    }
}
```

Example cont.

```
public class QuestionA
{
    public static void main(String args[] )
    {
        GenericStack<Integer> gs = new GenericStack<Integer>();
        gs.push(2);
        gs.push(1);
        System.out.println( gs.pop() );
    }
}
```

- Generic Type ‘Integer’
- Output 1

Summary

- Generic Types with Java
- Advantages of Generic Types
  - Flexibility/Robustness
- Incorporate Generic Types into your implementations
- Examples

This Week

- Read Associated Chapters
- Review Slides
- Java Exercises

Exercises

- Exercises 21.1 to 21.2 (Generics)

  25.1 to 25.2 (Arrays/Lists)

Questions/Discussion