# Revision

CS102A Lecture 16

James YU

Dec. 24, 2019

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Question 1

```java
public class A1 {
  public int x;
  private int y;
  protected int z;
  ...
}
```

```java
public class A2
    extends A1 {
  protected int a;
  private int b;
  ...
}
```

```java
public class A3
    extends A2 {
  private int q;
  ...
}
```

- Which of the following lists of instance data are accessible in class `A2`?
  1. `x`, `y`, `z`, `a`, `b`
  2. `x`, `y`, `z`, `a`
  3. `x`, `z`, `a`, `b`
  4. `z`, `a`, `b`
  5. `a`, `b`

# Question 1

```java
public class A1 {
  public int x;
  private int y;
  protected int z;
  ...
}
```

```java
public class A2
    extends A1 {
  protected int a;
  private int b;
  ...
}
```

```java
public class A3
    extends A2 {
  private int q;
  ...
}
```

- Which of the following lists of instance data are accessible in class A2?
  1. x, y, z, a, b
  2. x, y, z, a
  3. x, z, a, b
  4. z, a, b
  5. a, b
- x, z, a, b

# Question 2

```java
public class A1 {
  public int x;
  private int y;
  protected int z;
  ...
}
```

```java
public class A2
    extends A1 {
  protected int a;
  private int b;
  ...
}
```

```java
public class A3
    extends A2 {
  private int q;
  ...
}
```

- Which of the following lists of instance data are accessible in `A3`?
  1. `x`, `y`, `z`, `a`, `b`, `q`
  2. `a`, `b`, `q`
  3. `a`, `q`
  4. `x`, `z`, `a`, `q`
  5. `x`, `a`, `q`

# Question 2

```java
public class A1 {
  public int x;
  private int y;
  protected int z;
  ...
}
```

```java
public class A2
    extends A1 {
  protected int a;
  private int b;
  ...
}
```

```java
public class A3
    extends A2 {
  private int q;
  ...
}
```

- Which of the following lists of instance data are accessible in `A3`?
  1. `x`, `y`, `z`, `a`, `b`, `q`
  2. `a`, `b`, `q`
  3. `a`, `q`
  4. `x`, `z`, `a`, `q`
  5. `x`, `a`, `q`
- `x`, `z`, `a`, `q`

# Question 3

```java
public class A1 {
  public int x;
  private int y;
  protected int z;
  ...
}
```

```java
public class A2
    extends A1 {
  protected int a;
  private int b;
  ...
}
```

```java
public class A3
    extends A2 {
  private int q;
  ...
}
```

- Which of the following is true with the use of instance data `y` of class `A1`?
  1. it is accessible in `A1`, `A2` and `A3`
  2. it is accessible in `A1` and `A2`
  3. it is accessible only in `A1`
  4. it is accessible only in `A3`
  5. it is not accessible to any of the three classes

# Question 3

```java
public class A1 {
  public int x;
  private int y;
  protected int z;
  ...
}
```

```java
public class A2
    extends A1 {
  protected int a;
  private int b;
  ...
}
```

```java
public class A3
    extends A2 {
  private int q;
  ...
}
```

- Which of the following is true with the use of instance data `y` of class `A1`?
  1. it is accessible in `A1`, `A2` and `A3`
  2. it is accessible in `A1` and `A2`
  3. it is accessible only in `A1`
  4. it is accessible only in `A3`
  5. it is not accessible to any of the three classes
- it is accessible only in `A1`

# Question 4

- The instruction `super();` does which of the following?
  1. calls the method `super` as defined in the current class
  2. calls the method `super` as defined in the current class' parent class
  3. calls the method `super` as defined in java.lang
  4. calls the constructor as defined in the current class
  5. calls the constructor as defined in the current class' parent class

# Question 4

- The instruction `super();` does which of the following?
  1. calls the method `super` as defined in the current class
  2. calls the method `super` as defined in the current class' parent class
  3. calls the method `super` as defined in java.lang
  4. calls the constructor as defined in the current class
  5. calls the constructor as defined in the current class' parent class
- calls the constructor as defined in the current class' parent class

# Question 5

- Aside from permitting inheritance, the visibility modifier `protected` is also used to
  1. permit access to the protected item by any class defined in the same package
  2. permit access to the protected item by any `static` class
  3. permit access to the protected item by any parent class
  4. ensure that the class can not throw a `NullPointerException`
  5. define abstract elements of an `interface`

# Question 5

- Aside from permitting inheritance, the visibility modifier `protected` is also used to
  1. permit access to the protected item by any class defined in the same package
  2. permit access to the protected item by any `static` class
  3. permit access to the protected item by any parent class
  4. ensure that the class can not throw a `NullPointerException`
  5. define abstract elements of an `interface`
- permit access to the protected item by any class defined in the same package

# Question 6

- All classes in Java are directly or indirectly subclasses of the (  ) class.
  1. Wrapper
  2. String
  3. Reference
  4. this
  5. Object

# Question 6

- All classes in Java are directly or indirectly subclasses of the (  ) class.
  1. Wrapper
  2. String
  3. Reference
  4. this
  5. Object
- Object

# Question 7

- The reference to `getMoney()` in assignment 1 is to the class
  1. `Person`
  2. `Student`
  3. `Employee`
  4. `Retired`
  5. none of the above, this cannot be determined by examining the code

```
Person p = new Person(...);
// assignment 1
int m1 = p.getMoney();
p = new Student(...);
// assignment 2
int m2 = p.getMoney();
if (m2 < 100000)
  p = new Employee(...);
else if (m1 > 50000)
  p = new Retired(...);
// assignment 3
int m3 = p.getMoney();
```

# Question 7

- The reference to `getMoney()` in assignment 1 is to the class
  1. `Person`
  2. `Student`
  3. `Employee`
  4. `Retired`
  5. none of the above, this cannot be determined by examining the code
- `Person`

```
1  Person p = new Person(...);
2  // assignment 1
3  int m1 = p.getMoney();
4  p = new Student(...);
5  // assignment 2
6  int m2 = p.getMoney();
7  if (m2 < 100000)
8    p = new Employee(...);
9  else if (m1 > 50000)
10   p = new Retired(...);
11 // assignment 3
12 int m3 = p.getMoney();
```

# Question 8

- The reference to `getMoney()` in assignment 2 is to the class
  1. `Person`
  2. `Student`
  3. `Employee`
  4. `Retired`
  5. none of the above, this cannot be determined by examining the code

```
1  Person p = new Person(...);
2  // assignment 1
3  int m1 = p.getMoney();
4  p = new Student(...);
5  // assignment 2
6  int m2 = p.getMoney();
7  if (m2 < 100000)
8    p = new Employee(...);
9  else if (m1 > 50000)
10   p = new Retired(...);
11 // assignment 3
12 int m3 = p.getMoney();
```

# Question 8

- The reference to `getMoney()` in assignment 2 is to the class
  1. `Person`
  2. `Student`
  3. `Employee`
  4. `Retired`
  5. none of the above, this cannot be determined by examining the code
- `Student`

```
1  Person p = new Person(...);
2  // assignment 1
3  int m1 = p.getMoney();
4  p = new Student(...);
5  // assignment 2
6  int m2 = p.getMoney();
7  if (m2 < 100000)
8    p = new Employee(...);
9  else if (m1 > 50000)
10   p = new Retired(...);
11 // assignment 3
12 int m3 = p.getMoney();
```

# Question 9

- The reference to `getMoney()` in assignment 3 is to the class
  1. `Person`
  2. `Student`
  3. `Employee`
  4. `Retired`
  5. none of the above, this cannot be determined by examining the code

```
1  Person p = new Person(...);
2  // assignment 1
3  int m1 = p.getMoney();
4  p = new Student(...);
5  // assignment 2
6  int m2 = p.getMoney();
7  if (m2 < 100000)
8    p = new Employee(...);
9  else if (m1 > 50000)
10   p = new Retired(...);
11 // assignment 3
12 int m3 = p.getMoney();
```

# Question 9

- The reference to `getMoney()` in assignment 3 is to the class
  1. `Person`
  2. `Student`
  3. `Employee`
  4. `Retired`
  5. none of the above, this cannot be determined by examining the code
- none of the above, this cannot be determined by examining the code

```
1  Person p = new Person(...);
2  // assignment 1
3  int m1 = p.getMoney();
4  p = new Student(...);
5  // assignment 2
6  int m2 = p.getMoney();
7  if (m2 < 100000)
8    p = new Employee(...);
9  else if (m1 > 50000)
10   p = new Retired(...);
11 // assignment 3
12 int m3 = p.getMoney();
```

# Question 10

- The relationship between a child (sub) class and a parent (super) class is referred to as a(n) (  ) relationship.
  1. has-a
  2. is-a
  3. was-a
  4. instance-of

# Question 10

- The relationship between a child (sub) class and a parent (super) class is referred to as a(n) (  ) relationship.
  1. has-a
  2. is-a
  3. was-a
  4. instance-of
- is-a

# Question 11

```
1 public class Test {
2   private int x;
3   public test (int newValue) {
4     x = newValue;
5   }
6 }
```

- Which of the following is true about the class `Test`?
  1. it has no parent class
  2. it's parent class is `Object`
  3. it's parent class is Java
  4. it can not be extended
  5. it has a default child called `Object`

# Question 11

```java
public class Test {
  private int x;
  public test (int newValue) {
    x = newValue;
  }
}
```

- Which of the following is true about the class `Test`?
  1. it has no parent class
  2. it's parent class is `Object`
  3. it's parent class is Java
  4. it can not be extended
  5. it has a default child called `Object`
- it's parent class is `Object`

# Question 12

```
1 public class Test {
2   private int x;
3   public test (int newValue) {
4     x = newValue;
5   }
6 }
```

- If `q1` and `q2` are objects of `Test` class, then `q1.equals(q2)`
  1. is a syntax error since equals is not defined in the `Test` class
  2. is `true` if `q1` and `q2` both store the same value of `x`
  3. is `true` if `q1` and `q2` reference the same `Test` object
  4. is never `true`
  5. throws a `NullPointerException`

# Question 12

```
1 public class Test {
2   private int x;
3   public test (int newValue) {
4     x = newValue;
5   }
6 }
```

- If `q1` and `q2` are objects of `Test` class, then `q1.equals(q2)`
  1. is a syntax error since equals is not defined in the `Test` class
  2. is `true` if `q1` and `q2` both store the same value of `x`
  3. is `true` if `q1` and `q2` reference the same `Test` object
  4. is never `true`
  5. throws a `NullPointerException`
- is never `true`

# Question 13

```
1 for (int j = 0; j < list.length; j++)
2   if (list[j] < temp)
3     c++;
```

- What does the code do? Assume `list` is an initialized array of `int` values, `temp` is some previously initialized `int` value, and `c` is an `int` initialized to `0`.
  1. It finds the smallest value and stores it in `temp`
  2. It finds the largest value and stores it in `temp`
  3. It counts the number of elements equal to the smallest value in `list`
  4. It counts the number of elements in `list` that are less than `temp`
  5. It sorts the values in `list` to be in ascending order

# Question 13

```
1 for (int j = 0; j < list.length; j++)
2   if (list[j] < temp)
3     c++;
```

- What does the code do? Assume `list` is an initialized array of `int` values, `temp` is some previously initialized `int` value, and `c` is an `int` initialized to `0`.
  1. It finds the smallest value and stores it in `temp`
  2. It finds the largest value and stores it in `temp`
  3. It counts the number of elements equal to the smallest value in `list`
  4. It counts the number of elements in `list` that are less than `temp`
  5. It sorts the values in `list` to be in ascending order
- It counts the number of elements in `list` that are less than `temp`

# Question 14

```
try {
  BufferedReader f = new BufferedReader(new FileReader(filename)); //i1
  int x = Integer.parseInt(f.readLine()); //i2
  a[++i] = (double) (1 / x); //i3
}
catch (FileNotFoundException ex) {...} // e1
catch (NumberFormatException ex) {...} // e2
catch (ArithmeticException ex) {...} // e3
catch (ArrayIndexOutOfBounds ex) {...} // e4
catch (IOException ex) {...} // e5
```

- An exception raised by the instruction in `i1` would be caught by the `catch` statement labeled
  1) e1; 2) e2; 3) e5; 4) e1 or e5; 5) e1, e4 or e5

# Question 14

```
try {
  BufferedReader f = new BufferedReader(new FileReader(filename)); //i1
  int x = Integer.parseInt(f.readLine()); //i2
  a[++i] = (double) (1 / x); //i3
}
catch (FileNotFoundException ex) {...} // e1
catch (NumberFormatException ex) {...} // e2
catch (ArithmeticException ex) {...} // e3
catch (ArrayIndexOutOfBounds ex) {...} // e4
catch (IOException ex) {...} // e5
```

- An exception raised by the instruction in `i1` would be caught by the `catch` statement labeled
  1) e1; 2) e2; 3) e5; 4) e1 or e5; 5) e1, e4 or e5
- e1 or e5

# Question 15

```
try {
  BufferedReader f = new BufferedReader(new FileReader(filename)); //i1
  int x = Integer.parseInt(f.readLine()); //i2
  a[++i] = (double) (1 / x); //i3
}
catch (FileNotFoundException ex) {...} // e1
catch (NumberFormatException ex) {...} // e2
catch (ArithmeticException ex) {...} // e3
catch (ArrayIndexOutOfBounds ex) {...} // e4
catch (IOException ex) {...} // e5
```

- An exception raised by the instruction in `i2` would be caught by the `catch` statement labeled
  1) `e1`; 2) `e2`; 3) `e3`; 4) `e5`; 5) `e2` or `e5`

# Question 15

```
1  try {
2    BufferedReader f = new BufferedReader(new FileReader(filename)); //i1
3    int x = Integer.parseInt(f.readLine()); //i2
4    a[++i] = (double) (1 / x); //i3
5  }
6  catch (FileNotFoundException ex) {...} // e1
7  catch (NumberFormatException ex) {...} // e2
8  catch (ArithmeticException ex) {...} // e3
9  catch (ArrayIndexOutOfBounds ex) {...} // e4
10 catch (IOException ex) {...} // e5
```

- An exception raised by the instruction in `i2` would be caught by the `catch` statement labeled
  1) e1; 2) e2; 3) e3; 4) e5; 5) e2 or e5
- e2 or e5

# Question 16

```
try {
  BufferedReader f = new BufferedReader(new FileReader(filename)); //i1
  int x = Integer.parseInt(f.readLine()); //i2
  a[++i] = (double) (1 / x); //i3
}
catch (FileNotFoundException ex) {...} // e1
catch (NumberFormatException ex) {...} // e2
catch (ArithmeticException ex) {...} // e3
catch (ArrayIndexOutOfBounds ex) {...} // e4
catch (IOException ex) {...} // e5
```

- An exception raised by the instruction in `i3` would be caught by the `catch` statement labeled
  1) e2; 2) e3; 3) e4; 4) e3 or e4; 5) e2, e3 or e4

# Question 16

```
try {
  BufferedReader f = new BufferedReader(new FileReader(filename)); //i1
  int x = Integer.parseInt(f.readLine()); //i2
  a[++i] = (double) (1 / x); //i3
}
catch (FileNotFoundException ex) {...} // e1
catch (NumberFormatException ex) {...} // e2
catch (ArithmeticException ex) {...} // e3
catch (ArrayIndexOutOfBounds ex) {...} // e4
catch (IOException ex) {...} // e5
```

- An exception raised by the instruction in `i3` would be caught by the `catch` statement labeled
  1) e2; 2) e3; 3) e4; 4) e3 or e4; 5) e2, e3 or e4
- e3 or e4

# Question 17

- Consider the array declaration and instantiation: `int[] arr = new int[5];` Which of the following is `true` about `arr`?
  1. It stores 5 elements with legal indices between 1 and 5
  2. It stores 5 elements with legal indices between 0 and 4
  3. It stores 4 elements with legal indices between 1 and 4
  4. It stores 6 elements with legal indices between 0 and 5
  5. It stores 5 elements with legal indices between 0 and 5

# Question 17

- Consider the array declaration and instantiation: `int[] arr = new int[5];` Which of the following is `true` about `arr`?
  1. It stores 5 elements with legal indices between 1 and 5
  2. It stores 5 elements with legal indices between 0 and 4
  3. It stores 4 elements with legal indices between 1 and 4
  4. It stores 6 elements with legal indices between 0 and 5
  5. It stores 5 elements with legal indices between 0 and 5
- It stores 5 elements with legal indices between 0 and 4

# Question 18

- Which of the following messages passed to the string `str` could throw a `StringIndexOutOfBoundsException`?
  1. `str.length()`
  2. `str.charAt(2);`
  3. `str.replace('a', 'A');`
  4. `str.equals(str);`
  5. any of the above could throw a `StringIndexOutOfBoundsException`

# Question 18

- Which of the following messages passed to the string `str` could throw a `StringIndexOutOfBoundsException`?
  1. `str.length()`
  2. `str.charAt(2);`
  3. `str.replace('a', 'A');`
  4. `str.equals(str);`
  5. any of the above could throw a `StringIndexOutOfBoundsException`
- `str.charAt(2);`

# Question 19

- Assume `x` and `y` are `String` variables with `x = "Hello"` and `y = null`. If the operation `y = "Hello";` is performed, then the result of `(x == y)` is
  1. `true`
  2. `false`
  3. `x` and `y` becoming aliases
  4. `x` being set to the value `null`
  5. a run-time error

# Question 19

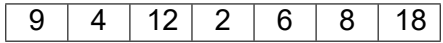
- Assume `x` and `y` are `String` variables with `x = "Hello"` and `y = null`. If the operation `y = "Hello";` is performed, then the result of `(x == y)` is
  1. `true`
  2. `false`
  3. `x` and `y` becoming aliases
  4. `x` being set to the value `null`
  5. a run-time error
- `false`

# Question 20

| 9 | 4 | 12 | 2 | 6 | 8 | 18 |
|---|---|---|---|---|---|---|

- What is returned by `values[1]`?

# Question 20




- What is returned by `values[1]`?
- `4`

# Question 21

| 9 | 4 | 12 | 2 | 6 | 8 | 18 |
|---|---|---|---|---|---|---|

- What is returned by `values.length`?

# Question 21

| 9 | 4 | 12 | 2 | 6 | 8 | 18 |
|---|---|---|---|---|---|---|

- What is returned by `values.length`?
- `7`

# Question 22

| 9 | 4 | 12 | 2 | 6 | 8 | 18 |
|---|---|---|---|---|---|---|

- Which of the following loops would adequately add 1 to each element stored in `values`?

1. `for (int j=1; j < values.length ; j++) values[j]++;`
2. `for (int j=0; j < values.length ; j++) values[j]++;`
3. `for (int j=0; j <= values.length; j++) values[j]++;`
4. `for (int j=0; j < values.length-1; j++) values[j]++;`
5. `for (int j=1; j < values.length-1; j++) values[j]++;`

# Question 22

| 9 | 4 | 12 | 2 | 6 | 8 | 18 |
|---|---|---|---|---|---|---|

- Which of the following loops would adequately add 1 to each element stored in `values`?
  1. `for (int j=1; j < values.length ; j++) values[j]++;`
  2. `for (int j=0; j < values.length ; j++) values[j]++;`
  3. `for (int j=0; j <= values.length; j++) values[j]++;`
  4. `for (int j=0; j < values.length-1; j++) values[j]++;`
  5. `for (int j=1; j < values.length-1; j++) values[j]++;`
- `for (int j=0; j < values.length ; j++) values[j]++;`

# Question 23

| 9 | 4 | 12 | 2 | 6 | 8 | 18 |
|---|---|---|---|---|---|---|

- The statement `System.out.println(values[7]);` will
  1. output `7`
  2. output `18`
  3. output nothing
  4. cause an `ArrayOutOfBoundsException` to be thrown
  5. cause a syntax error

# Question 23

| 9 | 4 | 12 | 2 | 6 | 8 | 18 |
|---|---|---|---|---|---|---|

- The statement `System.out.println(values[7]);` will
  1. output `7`
  2. output `18`
  3. output nothing
  4. cause an `ArrayOutOfBoundsException` to be thrown
  5. cause a syntax error
- cause an `ArrayOutOfBoundsException` to be thrown

# Question 24

- Consider that you want to extend `AClass` to `BClass`. `BClass` will have a third `int` instance data, `z`. Which of the following would best define `BClass`' constructor?

```
1 public BClass (int a, int b,
      int c) {
2   super(a, b);
3   z = c;
4 }
```

```
1  public class Aclass {
2    private int x;
3    protected int y;
4
5    public Aclass (int a, int b) {
6      x = a;
7      y = b;
8    }
9
10   public int addEm () {
11     return x + y;
12   }
13
14   public String toString () {
15     return "" + x + " " + y;
16   }
17 }
```

# Question 25

- Which of the following would best redefine the `toString` method for `BClass`?

```
public String toString () {
  return super.toString() + "
      " + z;
}
```

```
public class Aclass {
  private int x;
  protected int y;

  public Aclass (int a, int b) {
    x = a;
    y = b;
  }

  public int addEm () {
    return x + y;
  }

  public String toString () {
    return "" + x + " " + y;
  }
}
```