COMP 202

Review

Exam Format

- 10 true or false. (1 point each)
- 15 short answer. (2 points each)
- 1 long answer. (40 points)
- Total: 80 points

Exam Info

- Monday, June 19, 2017 from 9am to 12pm in ADAMS AUD (cap 332)
- Crib sheet: (up to) legal paper. Single or double sided.
- Hand-written or typed.
- Bring a pencil and an eraser!
- Please write your name as it appears on Minerva (not your preferred name). Write clearly.
- Sleep at least 6 hours in the 24 hours before the exam.

Exam Info

- Binary
- Control flow (if, loops (for, while))
- Primitive types
- Immutable reference types (String, Double, Integer, etc)
- Mutable reference types: arrays,
- Methods (input, returning, overloading)
- Exceptions (throw, throws, try, catch, finally)
- User-defined objects
  - public/private
  - Constructors
  - static vs non-static
  - Attributes
  - methods (incl. get/set, overloading, toString)
  - this

Not on Exam

- Recursion
- Generics (but you are free to use ArrayList)
- LinkedList.
- I/O Files.

Order of Operations

- Arithmetic, then comparisons, then boolean.
  1. parenthesis
  2. multiplication/division
  3. addition/subtraction

  1. relational <;<=, etc
  2. equality ==; ! =

  1. NOT
  2. AND
  3. OR
- Note: in the case of ties: they are performed from left to right!
- This is a good thing to put on your cheat sheet!

Order of Operations

```
String s = 1+2+3+"abc";
String t = "abc"+1+2+3;
String w = 1+2+3+"abc"+1+2+3;
```

- What are the characters in s?
- What are the characters in t?
- What are the characters in w?

Order of Operations

```
String s = 1+2+3+"abc";
String t = "abc"+1+2+3;
String w = 1+2+3+"abc"+1+2+3;
```

- What are the characters in s? //6abc
- What are the characters in t? //abc123
- What are the characters in w? //6abc123

Types

```
int x = 1/2;
double y = 3/4;
System.out.println(x+y+"abc");
System.out.println("abc"+x+y);
```

- What prints?

Types

```
int x = 1/2;
double y = 3/4;
System.out.println(x+y+"abc");
System.out.println("abc"+x+y);
```

- What prints?

  0.0abc

  abc00.0

Types

```
int x = 1/2;
double y = 3.0/4;
System.out.println(x+y+"abc");
System.out.println("abc"+x+y);
```

- What prints?

Types

```
int x = 1/2;
double y = 3.0/4;
System.out.println(x+y+"abc");
System.out.println("abc"+x+y);
```

- What prints?

  0.75abc

  abc00.75

Primitive Types

```
public static void main(String args[]) {
      int a = 5;
      method(a);
      System.out.println(a);
}
public static int method(int a){
      a++;
      return a;
}
```

- What prints?

Primitive Types

```
public static void main(String args[]) {
      int a = 5;
      method(a);
      System.out.println(a);
}
public static int method(int a){
      a++;
      return a;
}
```

- What prints?

  5

Mutable Reference Types

```
public static void main(String args[]) {
      int[] a = {1,2,3};
      method(a);
      System.out.println(a[0]);
}
public static void method(int[] b){
      for(int i=0; i<b.length; i++){
            b[i] = 10;
      }
}
```

- What prints?

Mutable Reference Types

```
public static void main(String args[]) {
      int[] a = {1,2,3};
      method(a);
      System.out.println(a[0]);
}
public static void method(int[] b){
      for(int i=0; i<b.length; i++){
            b[i] = 10;
      }
}
```

- What prints?

Mutable Reference Types

```
public static void main(String args[]) {
      int[] a = {1,2,3};
      method(a);
      System.out.println(a[0]);
}
public static void method(int[] b){
      b = new int[5];
      for(int i=0; i<b.length; i++){
            b[i] = 10;
      }
}
```

- What prints?

Mutable Reference Types

```
public static void main(String args[]) {
      int[] a = {1,2,3};
      method(a);
      System.out.println(a[0]);
}
public static void method(int[] b){
      b = new int[5];
      for(int i=0; i<b.length; i++){
            b[i] = 10;
      }
}
```

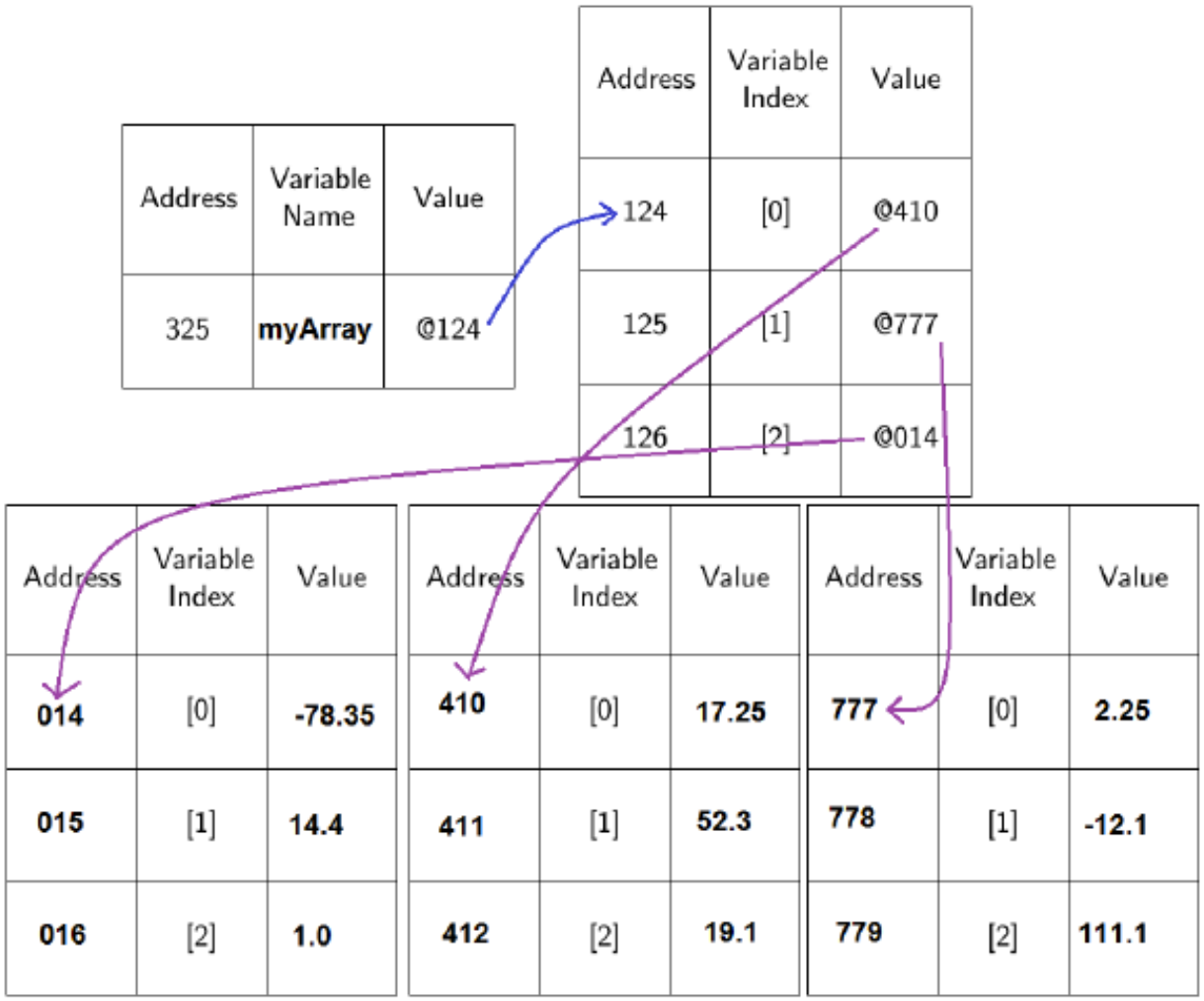
- What prints?

  1

Arrays of Arrays

- References to References:
-
```
double[][] myArray = {{17.25, 52.3, 19.1}};
                      {2.25, -12.1, 111.1}
                      {-78.35, 14.4, 1.0}};
```

- `double[][] myArray = {{17.25, 52.3, 19.1}};`
  `{2.25, -12.1, 111.1}`
  `{-78.35, 14.4, 1.0}};`



Swap with Arrays

```
public static void main(String[] args){
	int[][] a = {{1,2},{3,4}};
	swap   (a, 0, 1);
	System.out.println(Arrays.deepToString(a));
}
public static void swap(int[][] b, int i,int j){
	int temp = b[i][j];
	b[i][j] = b[j][i];
	b[j][i] = temp;
}
```

- What prints?

Swap with Arrays

```
public static void main(String[] args){
	int[][] a = {{1,2},{3,4}};
	swap   (a, 0, 1);
	System.out.println(Arrays.deepToString(a));
}
public static void swap(int[][] b, int i,int j){
	int temp = b[i][j];
	b[i][j] = b[j][i];
	b[j][i] = temp;
}
```

- What prints?

  [[1, 3], [2, 4]]

Immutable Reference Types

```
public static void main(String[] args){
      String s = "abc";
      method(s);
      System.out.println(s);
}
public static void method(String t){
      t = t + "123";
}
```

- What prints?

Immutable Reference Types

```
public static void main(String[] args){
      String s = "abc";
      method(s);
      System.out.println(s);
}
public static void method(String t){
      t = t + "123";
}
```

- What prints?

  abc

Method overloading

- A method in Java is overloaded if there is more than one method with the same name in the same class.
- In order to do this, you must give the method dierent inputs.
- When an overloaded method is called, the compiler matches on the input types to figure out which version to use. It picks the closest/best valid match.

Overloading

```
public static void method(int x, int y)
```

- Which of the following are valid ways to overload the above method?

1. `public int method(int x, int y)`
2. `public static void method(double x, double y)`
3. `private void method(int x)`
4. `public static void method(String x, String y)`
5. `public static void method(int a, int b)`

User-Defined Objects

- A constructor is a method that is called automatically when you create an instance of a class using the `new` keyword. It has no return type and is required to have the same name as the name of the class.
- An object is created every time the `new` keyword is used.

private vs public

- An attribute (or method) that is declared to be public can be directly accessed from outside of the class file in which it was declared.
- An attribute (or method) that is declared to be private can only be directly accessed from inside of the class file in which it was declared.

get/set methods

- If you would like to allow for a user to access or modify the value of a private class attribute, you can do this by writing get (and/or) set methods for this attribute.

mutable/immutable

- If you define a class with only private attributes and no set methods, then you have essentially defined an immutable object.
- There is no way to changes the contents of the object after it has been created.

static and this

- A method or attribute that is declared to be `static` belongs to the whole class. It never makes sense to use the `this` keyword inside of a static method.
- A non-static method or attribute belongs to an instance of a class.
- It is always reasonable to use the `this` keyword inside of a non-static method.
- The `this` keyword is used to directly refer to the object on which a non-static method was called.

static or non-static?

- (Assume the methods are called from the correct context)
  - `myArrayList.size()`
  - `Math.pow(2,5)`
  - `s.charAt(3)`
  - `int x = Integer.parseInt(\12")`
  - `boolean b = a1.equals(a2)`
  - `int x = input.nextInt()`
  - `String s = String.valueOf(3.1415)`

static or non-static?

- Assume we have a class that denes a Cat. Consider the following attributes that might be included:
- The name of a cat
  - `c.name;`
- The kingdom (in the biological taxonomy sense) of a cat
  - `Cat.kingdom;`
- The number of chromosomes
  - `Cat.numChromosomes;`
- The age of a cat
  - `c.age;`

Objects and Reference Types

```
public class Student{
	private String name;
	private long[] studentIds;
	public Student(String name, long[] ids){
		this.name = name;
		this.studentIds = ids;
	}
	public static void main(String[] args){
		long[] nums = {12345678, 87654321};
		Student s = new Student("Hogwarts",nums);
		nums[0] = 1;
		System.out.println(s.studentIds[0]);
		nums = new int[5];
		System.out.println(s.studentIds.length);
	}
} // What prints?
```

Objects and Reference Types

```
public class Student{
        private String name;
        private long[] studentIds;
        public Student(String name, long[] ids){
                this.name = name;
                this.studentIds = ids;
        }
        public static void main(String[] args){
                long[] nums = {12345678, 87654321};
                Student s = new Student("Hogwarts",nums);
                nums[0] = 1;
                System.out.println(s.studentIds[0]);
                nums = new int[5];
                System.out.println(s.studentIds.length);
        }
}  // What prints?
        Compiler error => Type mismatch
```

Objects and Reference Types

```
public class Student{
	private String name;
	private long[] studentIds;
	public Student(String name, long[] ids){
		this.name = name;
		this.studentIds = ids;
	}
	public static void main(String[] args){
		long[] nums = {12345678, 87654321};
		Student s = new Student("Hogwarts",nums);
		nums[0] = 1;
		System.out.println(s.studentIds[0]);
		nums = new long[5];
		System.out.println(s.studentIds.length);
	}
} // What prints?
```

Objects and Reference Types

```
public class Student{
        private String name;
        private long[] studentIds;
        public Student(String name, long[] ids){
                this.name = name;
                this.studentIds = ids;
        }
        public static void main(String[] args){
                long[] nums = {12345678, 87654321};
                Student s = new Student("Hogwarts",nums);
                nums[0] = 1;
                System.out.println(s.studentIds[0]);
                nums = new long[5];
                System.out.println(s.studentIds.length);
        }
}  // What prints?
```

1
2

Objects and Reference Types

```
public class Student{
	private String name;
	private long[] studentIds;
	public Student(String name, long[] ids){
		this.name = name;
		this.studentIds = new
long[this.studentIds.length];
	}
	public static void main(String[] args){
		long[] nums = {12345678, 87654321};
		Student s = new Student("Hogwarts",nums);
		nums[0] = 1;
		System.out.println(s.studentIds[0]);
	}
}  // What prints?
```

Objects and Reference Types

```
public class Student{
	private String name;
	private long[] studentIds;
	public Student(String name, long[] ids){
		this.name = name;
		this.studentIds = new
long[this.studentIds.length];
	}
	public static void main(String[] args){
		long[] nums = {12345678, 87654321};
		Student s = new Student("Hogwarts",nums);
		nums[0] = 1;
		System.out.println(s.studentIds[0]);
	}
} // What prints?
```

NullPointerException

Objects and Reference Types

```
public class Student{
	private String name;
	private long[] studentIds;
	public Student(String name, long[] ids){
		this.name = name;
		this.studentIds = new  long[1];
	}
	public static void main(String[] args){
		long[] nums = {12345678, 87654321};
		Student s = new Student("Hogwarts",nums);
		nums[0] = 1;
		System.out.println(s.studentIds[0]);
	}
} // What prints?
```

Objects and Reference Types

```
public class Student{
	private String name;
	private long[] studentIds;
	public Student(String name, long[] ids){
		this.name = name;
		this.studentIds = new  long[1];
	}
	public static void main(String[] args){
		long[] nums = {12345678, 87654321};
		Student s = new Student("Hogwarts",nums);
		nums[0] = 1;
		System.out.println(s.studentIds[0]);
	}
}
```

// What prints?

0

Objects and Reference Types

```
public class Student{
	private String name;
	private long[] studentIds;
	public Student(String name, long[] ids){
		this.name = name;
		this.studentIds = new  long[0];
	}
	public static void main(String[] args){
		long[] nums = {12345678, 87654321};
		Student s = new Student("Hogwarts",nums);
		nums[0] = 1;
		System.out.println(s.studentIds[0]);
	}
} // What prints?
```

Objects and Reference Types

```
public class Student{
	private String name;
	private long[] studentIds;
	public Student(String name, long[] ids){
		this.name = name;
		this.studentIds = new  long[0];
	}
	public static void main(String[] args){
		long[] nums = {12345678, 87654321};
		Student s = new Student("Hogwarts",nums);
		nums[0] = 1;
		System.out.println(s.studentIds[0]);
	}
} // What prints?
```

ArrayIndexOutOfBoundsException

Objects and Reference Types

```
public class Student{
	private String name;
	private long[] studentIds;
	public Student(String name, long[] ids){
		this.name = name;
		this.studentIds = new long[ids.length];
		studentIds[0] = ids[0];
		studentIds[1] = ids[1];  }
	public static void main(String[] args){
		long[] nums = {12345678, 87654321};
		Student s = new Student("Hogwarts",nums);
		nums[0] = 1;
		System.out.println(s.studentIds[0]);
	}
}  // What prints?
```

Objects and Reference Types

```
public class Student{
	private String name;
	private long[] studentIds;
	public Student(String name, long[] ids){
		this.name = name;
		this.studentIds = new long[ids.length];
		studentIds[0] = ids[0];
		studentIds[1] = ids[1];  }
	public static void main(String[] args){
		long[] nums = {12345678, 87654321};
		Student s = new Student("Hogwarts",nums);
		nums[0] = 1;
		System.out.println(s.studentIds[0]);
	}
}  // What prints?
	12345678
```

Exceptions

- If you would like to flag an illegal operation in your code, you can explicitly throw Exception.

```
public static double mySqrt(double x){
      if(x<0){
            throw new IllegalArgumentException(
            "Cannot take the square root of a
negative number");
      }
      ...
}
```

Exceptions

- There are two ways to handle a checked Exception:
  1. Put throws Exception in the method header.
  2. Put your code in a try block.

Option 1

```
public static int method() throws IOException{
	.. some code here
}
public static void main(String[] args){
	try{
		int x = method();
	}catch(IOException e){
		e.printStackTrace();
	}
}
```

Catching Exceptions

In order to handle exceptions, you can catch them as follows:

```
try {
      \\potentially problematic code here
}
catch(Exception e) {
      \\problem resolution code here
}
      \\The code continues to run.
```

Recall that any type of Exception will be caught by a general catch `Exception e`.

Catching Multiple Exceptions

- If you would like to do something different depending on the type of Exception caught, you can use multiple catch blocks.

```
try {
      \\potentially problematic code here
}
catch(NullPointerException e) {
      \\problem resolution 1 code here
}
catch(Exception e) {
      \\problem resolution 2 code here
}
\\The code can continue to run
```

- A try block will always match with at most one Exception in the catch sequence.

If you always want to do something

The `finally` block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. You can have a `finally` with just a try (and no catch). A try-finally does not handle a checked Exception.

```
try {
      \\potentially problematic code here
}
catch(NullPointerException e) {
      \\problem resolution 1 code here
}
finally {
      \\This always happens
}
\\The code continues to run.
```

What Prints?

```
public static void main(String[] args){
  int[] p = new int[5];
  try {
       for(int i=0; i<=p.length; i++){
            System.out.println(p[i]);
       }
  }
  catch(NullPointerException e) {
       System.out.println("Null pointer!");
  }
  catch(ArrayIndexOutOfBoundsException e) {
       System.out.println("Bad Index!");
  }
  System.out.println("Moving on");
}
```

What Prints?

```
public static void main(String[] args){
  int[] p = new int[5];
  try {
       for(int i=0; i<=p.length; i++){
             System.out.print(p[i]);
       }
  }
  catch(NullPointerException e) {
       System.out.print("Null pointer!");
  }
  catch(ArrayIndexOutOfBoundsException e) {
       System.out.print("Bad Index!");
  }
  System.out.print("Moving on");
}
```

00000Bad Index!Moving on

What Prints?

```
public static void main(String[] args){
  Integer[] p = new Integer[5];
  try {
       for(int i=0; i<=p.length; i++){
            System.out.print(p[i].intValue());
       }
  }
  catch(NullPointerException e) {
       System.out.print("Null pointer!");
  }
  catch(ArrayIndexOutOfBoundsException e) {
       System.out.print("Bad Index!");
  }
  System.out.print("Moving on");
}
```

What Prints?

```
public static void main(String[] args){
   Integer[] p = new Integer[5];
   try {
        for(int i=0; i<=p.length; i++){
              System.out.print(p[i].intValue());
        }
   }
   catch(NullPointerException e) {
        System.out.print("Null pointer!");
   }
   catch(ArrayIndexOutOfBoundsException e) {
        System.out.print("Bad Index!");
   }
   System.out.print("Moving on");
}
```

- Null pointer!Moving on

What Prints?

```
public static void main(String[] args){
    int[] a = new int[0];
    try {
        a[0] = 12;
    }
    catch(NullPointerException e) {
        System.out.println("Null pointer!");
    }
    finally {
        System.out.println("Here!");
    }
        System.out.println("Moving on");
}
```

What Prints?

```
public static void main(String[] args){
    int[] a = new int[0];
    try {
        a[0] = 12;
    }
    catch(NullPointerException e) {
        System.out.println("Null pointer!");
    }
    finally {
        System.out.println("Here!");
    }
        System.out.println("Moving on");
}
```

Here! ArrayIndexOutOfBoundsException

What Prints?

```
public static void main(String[] args){
    int[] a = new int[0];
    try {
        a[0] = 12;
    }
    System.out.println(“Hello”);
    catch(NullPointerException e) {
        System.out.println("Null pointer!");
    }
    finally {
        System.out.println("Here!");
    }
        System.out.println("Moving on");
}
```

What Prints?

```
public static void main(String[] args){
    int[] a = new int[0];
    try {
        a[0] = 12;
    }
    System.out.println(“Hello”);
    catch(NullPointerException e) {
        System.out.println("Null pointer!");
    }
    finally {
        System.out.println("Here!");
    }
        System.out.println("Moving on");
}
Compiler time error.
```