Introduction to java

COMP1030
Lecture #6

Housekeeping

- Our goal - 100% pass rate in this course.
- Review the lecture slides ahead of time.
- Review the lecture slides after class with a study group.
- Repeat the lab at home 1-2 times.
- Take notes during class.
- Weekly optional tutorials take place every week on Wednesdays, from 3-4pm in room N111 (note new location.

Housekeeping

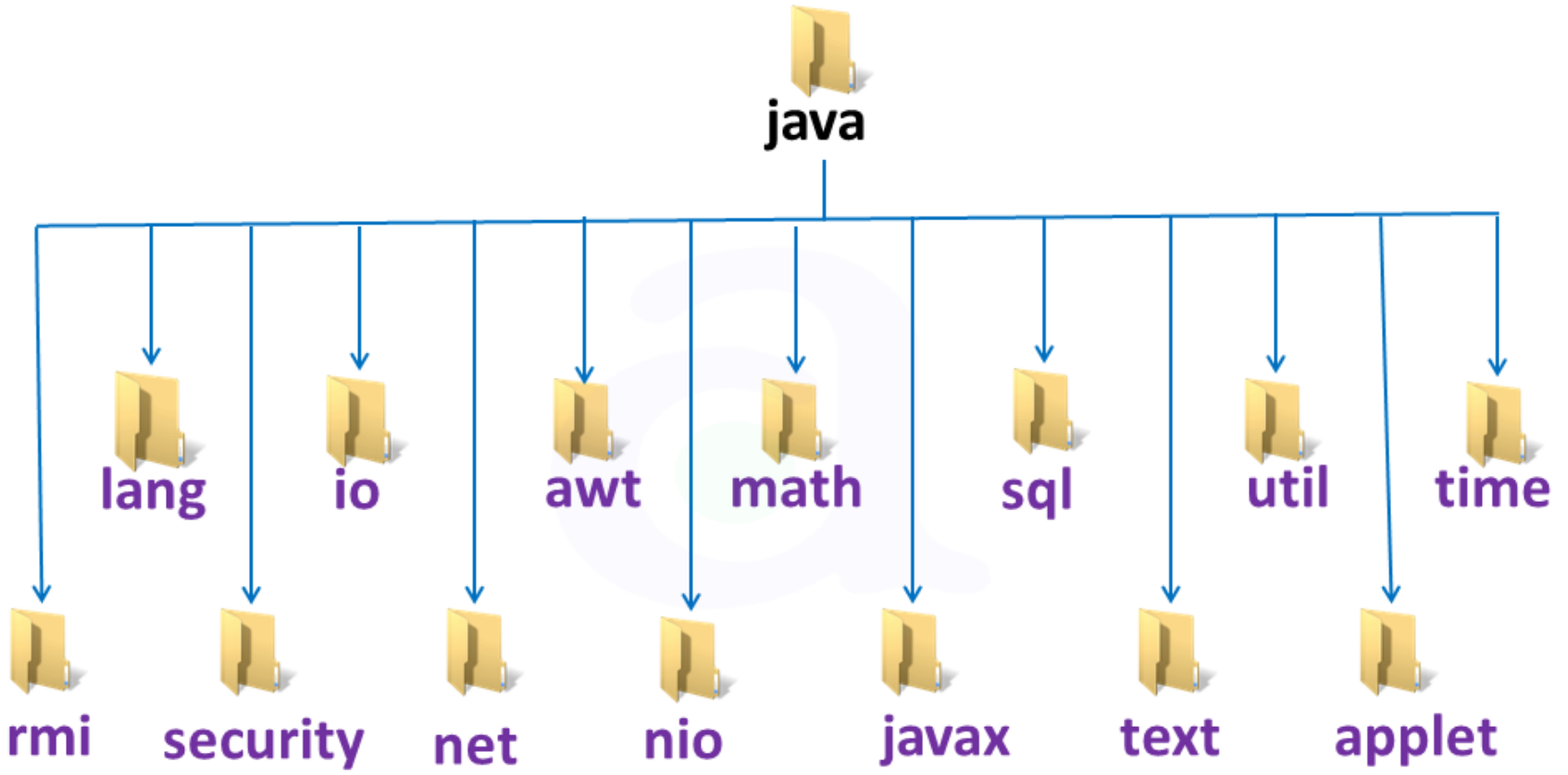
- Assignment #1 Answer key is posted under the assignments link
- Misterm-Exam question paper and answer key are posted under course information link
- Mid-term exams can be photographed during todays lab
- Labs will no longer be checked as homework
- Students will be expected to work in a lab group – be prepared to change seats for lab

Review

- for statement
- do-while statement
- switch statement

Software re-usability

Java packages



Software re-usability

- Use existing classes from existing java packages to reduce development time and programming errors
- Dividing a program into meaningful classes and methods avoids re-writing code and supports easier debugging and maintenance

Software re-usability

- Every method should be limited to performing a single well defined task.

- The name of the method should express its task.

Static methods and fields

- Called class methods
- Do not require an object to be invoked
- To declare a method as static place the keyword static before the return type in the methods declaration

Static methods and fields

- A static method can be called (invoked) using the class name and the dot operator:

  ClassName.methodName (arguments)

Static methods and fields

- The Math class provides several convenient static methods

```
Math.sqrt(900);
```

Static methods and fields

Some Math class methods

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.ceil(`*value*`)`	rounds up
`Math.floor(`*value*`)`	rounds down
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1*`, `*value2*`)`	larger of two values
`Math.min(`*value1*`, `*value2*`)`	smaller of two values
`Math.pow(`*base*`, `*exp*`)`	*base* to the *exp* power
`Math.random()`	random `double` between 0 and 1
`Math.round(`*value*`)`	nearest whole number
`Math.sqrt(`*value*`)`	square root

`Math.sin(`*value*`)`
`Math.cos(`*value*`)`
`Math.tan(`*value*`)`
	sine/cosine/tangent of an angle in radians

`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

Static methods and fields

Math.max (x,y);

Math.max(32,45);

Math.max(x, Math.max(y,z));

this line of code finds the largest of three values

Static methods and fields

- Static variables are also known as class variables.
- All objects share **one** copy of static variables.
- PI for example is declared: *public static final*
- A final variable cannot be changed once it is initialized.

Concatenation

- The + operator can be used to assemble String objects into larger Strings

- The + operator is a good illustration of polymorphism

"hello" + "world" --- + means concatenation

3 +4 --- + means plus

Concatenation

- System.out.println("Maximum is: " + result);
- Assuming result was an int, result was converted to a String by java and then concatenated to "Maximum is: "
- Java evaluates the operands of an operator from left to right

Concatenation

- Example

```
int y = 5;
System.out.println("y+2=" + y + 2);
```

prints: y+2=52

why?

Concatenation

- Example

```
int y = 5;
System.out.println("y+2=" + (y + 2));
```

prints: y+2=7

why?

Method invocation

There are three ways to invoke a method:

- A method in the same class can be called with just the method name. (no reference or dot operator)
- Using a reference to the object and the dot operator. (instance method)
- Using the class name and the dot operator. (static method)

Method compiler errors

- Static methods are typically not used to access instance variables or instance methods of the same class because an object may not yet be created.
- Declaring a method outside the body of a class or inside the body of another method will result in a syntax error

Method Call Stack

- A stack is a data structure or collection of related data items.

Method Call Stack

- When a plate (data) is placed on top of the stack it is referred to as “push”.

- When a plate (data) is removed from the stack it is known as “pop”.
- Most stacks are based upon a LIFO architecture.

Method Call Stack

- When a program calls a method, program execution leave its current location and "jumps" to another location in the code.

- Once the method is completed, the program needs to know where it left off. The memory location of where it is left off is stored on the stack.

Method Call Stack

- If multiple method calls are made in succession, the successive return addresses are pushed onto the stack in LIFO order.

- However, the computers memory is finite (recall modulus arithmetic), therefore a finite number of successive method calls can be accommodated. Too many method calls will cause a stack overflow error.

Argument Promoting and Casting

- The Math.sqrt() method expects a double argument.
- However Math.sqrt(4); is valid because java will automatically promote (convert) the int 4 to a double because no data will be lost in this promotion.

Argument Promoting and Casting

Promotions allowed for primitive types

Type	Valid promotions
double	None
float	double
long	float or double
int	long, float or double
char	int, long, float or double
short	int, long, float or double
byte	short, int, long, float or double
boolean	None (boolean values are not considered to be numbers in Java)

Argument Promoting and Casting

- Converting a double to an int will truncate the fractional portion of the double value, therefore part of the value is lost.
- If one attempts to use a type that requires a demotion (converting to a lower type) the compiler will error out.
- To force conversion (essentially saying to the compiler, I know this conversion might cause loss of information, but in this situation it is ok) we can use the cast operator.

Argument Promoting and Casting

Example

- A method called calculateResult requires an argument of type int.

- To pass a double we would need to cast the double to an int: (this is known as type casting)

```
ref.calculateResult((int)32.64);
```

Scope

- The scope of a variable is the portion of the program that can refer to the declared variable by its name.
- If a variable can be referred to by its name it is said to be in scope for that portion of the program.

General Scope Rules

- The scope of a parameter declaration is the body of the method in which the declaration appears.
- The scope of a local variable declaration is from the point at which the declaration appears to the end of that block.

General Scope Rules

- The scope of a local variable declaration that appears in the initialization section of a statement header is the body of the for statement and the other expressions in the header.
- A method or instance variable scope is the entire body of the class.

Method Overloading

- Methods of the same name can be declared in the same class, as long as they have a different signature relative to the parameters.
  - # of parameters
  - Types of parameters
  - Order of parameters

Method Overloading

- When the overloaded method is invoked, the compiler selects the appropriate method based upon the parameter list.
- Example: the Math.max() method has 4 overloaded versions which take as parameters:
  - Two doubles
  - Two floats
  - Two longs
  - Two ints
- A different return type does **not** constitute an overloaded method.

Method Overloading

```
3   public class OverloadingEx {
4       void sum(int a, int b) {
5           int sum = a + b;
6           System.out.println("sum of no is " + sum);
7
8       }
9
10      void sum(double a, double b) {
11          double sum = a + b;
12          System.out.println("sum of no is " + sum);
13      }
14
```