Programming in Java: lecture 5

- Return Values
- APIs, Packages and Javadoc
- More on Program Design
- Declarations
- Something about learning
- Repetition

Slides made for use with ”Introuction to Programming Using Java, Version 5.0” by David J. Eck
Some figures are taken from ”Introuction to Programming Using Java, Version 5.0” by David J. Eck
Lecture 3 covers Section 4.4 to 4.7 + some repitition

Lecture 4

- Exceptions and try...catch
- Overview – static vs. non static
- GUI programming – Applets
- Black Boxes
  - Subroutines
  - Local and Global variables
  - Parameters – formal and actual
  - Overloading

Exceptions

- Lecture 3: Normal flow of control
  - Why do we need something different
  - Handle errors somewhere else then where they happen
- Exception – the exception is an Object
- try...catch statements

try...catch

- Formal syntax

```
try {
   ⟨statements-1⟩
}
catch ( ⟨exception-class-name⟩  ⟨variable-name⟩ ) {
   ⟨statements-2⟩
}
```

try...catch

- Example

```
try {
   double x;
   x = Double.parseDouble(str);
   System.out.println( "The number is " + x );
}
catch ( NumberFormatException e ) {
   System.out.println( "Not a legal number." );
}
```

Bad parameter values

- This is an error
- What do you do?
- Throw an exception

```
static void print3NSequence(int startingValue) {

   if (startingValue <= 0)  // The contract is violated!
      throw new IllegalArgumentException( "Starting value must be positive." );

   .
   .  // (The rest of the subroutine is the same as before.)
   .
```

Return Values

- Function – subroutines with a return value
  - Can only return one specific type
- Can be used as expressions or statements
  - Statement: return value is ignored
  - Test condition: boolean value

The return statement

- `return <expression>;`
- Should give some result of the same type as the return value of the function
- Must be inside function
- Example

```
static double pythagoras(double x, double y) {
      // Computes the length of the hypotenuse of a right
      // triangle, where the sides of the triangle are x and y.
    return  Math.sqrt( x*x + y*y );
}
```

Function Examples

- The 3N+1 Sequence

```
static int nextN(int currentN) {
   if (currentN % 2 == 1)     // test if current N is odd
      return 3*currentN + 1;  // if so, return this value
   else
      return currentN / 2;    // if not, return this instead
}
```

- return type void.
  - `return;`

One return statement

- Some people prefer having only one return statement per function

```
static int nextN(int currentN) {
   int answer;  // answer will be the value returned
   if (currentN % 2 == 1)    // test if current N is odd
      answer = 3*currentN+1; // if so, this is the answer
   else
      answer = currentN / 2; // if not, this is the answer
   return answer;   // (Don't forget to return the answer!)
}
```

Use of Functions

```
static void print3NSequence(int startingValue) {

   int N;       // One of the terms in the sequence.
   int count;   // The number of terms found.

   N = startingValue;   // Start the sequence with startingValue.
   count = 1;

   TextIO.putln("The 3N+1 sequence starting from " + N);
   TextIO.putln();
   TextIO.putln(N);  // print initial term of sequence

   while (N > 1) {
       N = nextN( N );   // Compute next term, using the function nextN.
       count++;          // Count this term.
       TextIO.putln(N);  // Print this term.
   }

   TextIO.putln();
   TextIO.putln("There were " + count + " terms in the sequence.");

}
```

Return type can be any type

- `static boolean isPrime(int N);`
- `static String reverse(String str);`

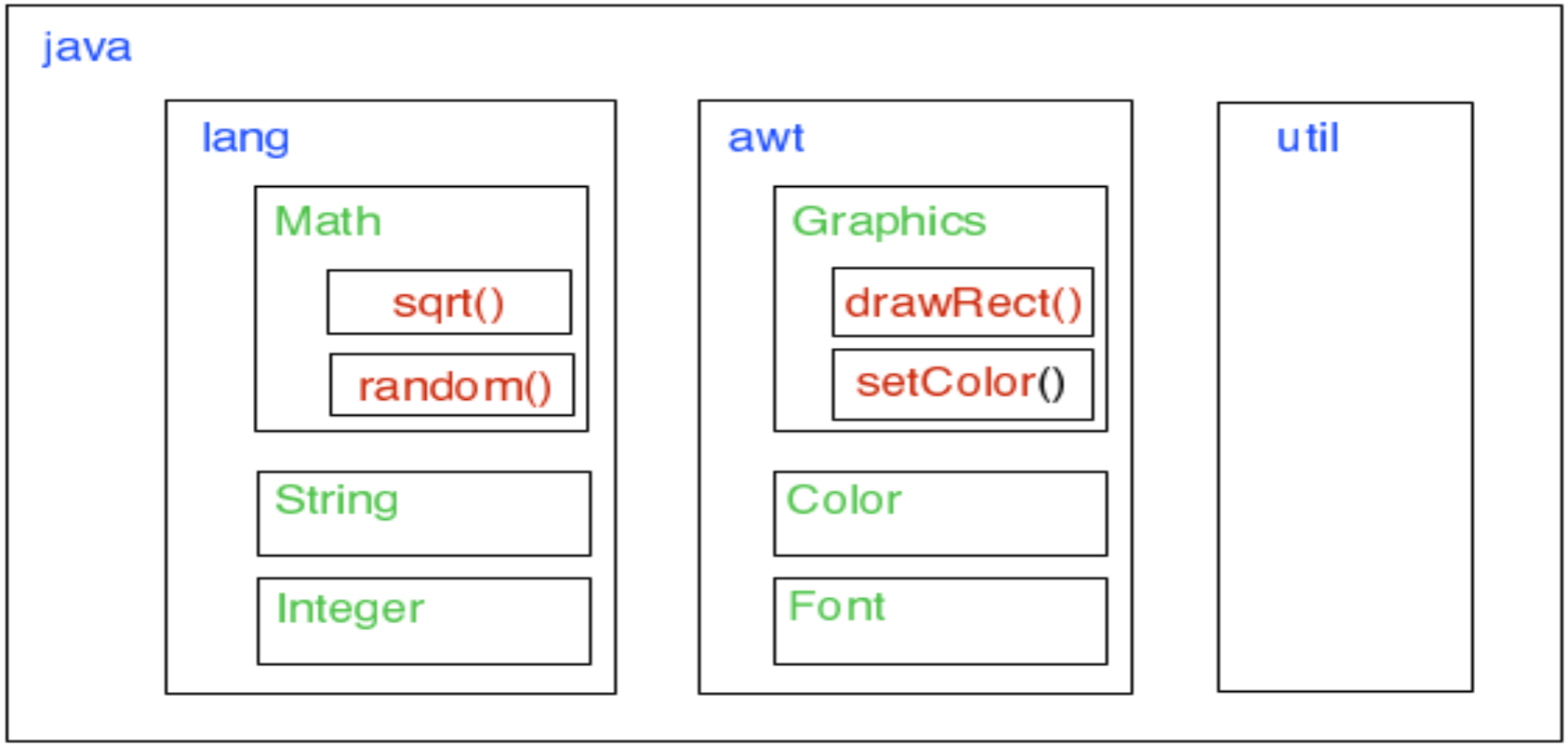
```
static String reverse(String str) {
   String copy;  // The reversed copy.
   int i;        // One of the positions in str,
                 //       from str.length() - 1 down to 0.
   copy = "";    // Start with an empty string.
   for ( i = str.length() - 1;  i >= 0;  i-- ) {

            // Append i-th char of str to copy.
      copy = copy + str.charAt(i);
   }
   return copy;
}
```

APIs, Packages and Javadoc

- API – Application Programmers Interface
  - What you need to know from the outside
  - Windows, MacOS, linux (gtk, gnome), Java
  - Math Toolboxes

Packages

- Too much functionality to expose it all at once



Subroutines nested in classes nested in two layers of packages.
The full name of sqrt() is java.lang.Math.sqrt()

Import directives

- Technically not a statement
  - java.lang.*; // automatically imported, contains String
- Import java.*
  - does not import everything
- GUI program: typical import
  - import java.awt.*;
  - import java.awt.event.*; // still needed
  - import javax.swing.*;
- Javax is additions from java 1.2

Name conflicts

- Two classes in different packages with the same name
- java.awt.List
- java.util.List
- Only importing specific packages or
- Using fully qualified names

Create your own package

- Eclipse warns about using the default package
- Packages are stored in Java Archives
  - .jar files

Javadoc

- Comments used for generating documentation
- Begins with /**

```
/**
 * This subroutine prints a 3N+1 sequence to standard output, using
 * startingValue as the initial value of N.  It also prints the number
 * of terms in the sequence. The value of the parameter, startingValue,
 * must  be a positive integer.
 */

static void print3NSequence(int startingValue) { ...
```

Semantic description

- Syntactic information in function name, return type and argument types
- Javadoc can contain HTML code
- doc tags

```
@param  ⟨parameter-name⟩  ⟨description-of-parameter⟩

@return  ⟨description-of-return-value⟩

@throws  ⟨exception-class-name⟩  ⟨description-of-exception⟩
```

Example

```
/**
 * This subroutine computes the area of a rectangle, given its width
 * and its height.  The length and the width should be positive numbers.
 * @param width the length of one side of the rectangle
 * @param height the length the second side of the rectangle
 * @return the area of the rectangle
 * @throws IllegalArgumentException if either the width or the height
 *    is a negative number.
 */
public static double areaOfRectangle( double length, double width ) {
    if ( width < 0  ||  height < 0 )
       throw new IllegalArgumentException("Sides must have positive length.");
    double area;
    area = width * height;
    return area;
}
```

More on Program Design

- Preconditions and
- Postcondition

```
/**
 * Sets the color of one of the rectangles in the window.
 *
 * Precondition:   row and col are in the valid range of row and column numbers,
 *                   and r, g, and b are in the range 0 to 255, inclusive.
 * Postcondition:  The color of the rectangle in row number row and column
 *                   number col has been set to the color specified by r, g,
 *                   and b.  r gives the amount of red in the color with 0
 *                   representing no red and 255 representing the maximum
 *                   possible amount of red.  The larger the value of r, the
 *                   more red in the color.  g and b work similarly for the
 *                   green and blue color components.
 */
public static void setColor(int row, int col, int r, int g, int b)
```

Declarations

- Initialization in declarations

```
int count;     // Declare a variable named count.
count = 0;     // Give count its initial value.
```

- is the same as

```
int count = 0;  // Declare count and give it an initial value.
```

- Multiple initializations

```
char firstInitial = 'D', secondInitial = 'E';

int x, y = 1;    // OK, but only y has been initialized!

int N = 3, M = N+2;  // OK, N is initialized
                     //        before its value is used.
```

For loops

- Initialization in for loops

```
for ( int i = 0;  i < 10;  i++ ) {
   System.out.println(i);
}
```

- is the same as

```
{
    int i;
    for ( i = 0;  i < 10;  i++ ) {
       System.out.println(i);
    }
}
```

Static member variables

- Can be initialized when declared

```
public class Bank {
    static double interestRate = 0.05;
    static int maxWithdrawal = 200;
```

- No statements outside functions

```
public class Bank {
   static double interestRate;
   interestRate = 0.05;  // ILLEGAL:
   .                     //    Can't be outside a subroutine!:
   .
```

Named Constants

- Can easily be changed between compiles
- `final static double interestRate = 0.05;`
- `final static double INTEREST_RATE = 0.05;`
- `Math.PI;`
- Enumerated type constants
- `Color.RED`

Naming and Scope Rules

- Scope – Hvad man kan se
- Member variables are in scope in the Class
- Hiding outer variable with the same name
- Game.count to get member variable

```
public class Game {

    static int count;  // member variable

    static void playGame() {
        int count;  // local variable
            .
            .   // Some statements to define playGame()
            .
    }
```

Only one level of nesting

- You can only have one level of nesting of variables with the same name

```
void  badSub(int y) {
    int x;
    while (y > 0) {
        int x;  // ERROR:  x is already defined.
            .
            .
            .
    }
}
```

- Ok with multiple on the same level

Insanity

- `static Insanity Insanity( Insanity Insanity) { ... }`
- Do not do this!
- Remember the pragmatics

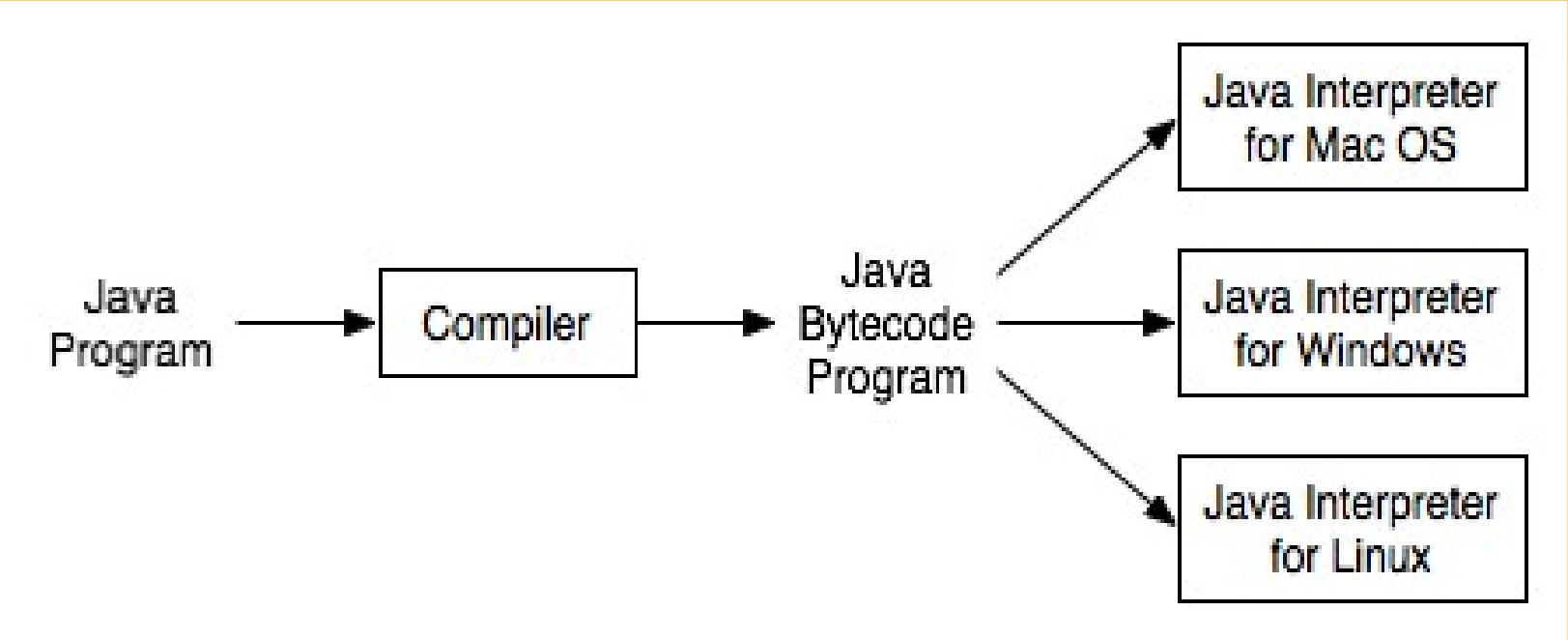
Something about learning

- Repetition – teaches your brain to remember
- Programming is an activity not facts
- Doing is learning
- You should be able to do the exercises
- Watching others do the exercises will not teach you much

Repetition

Java Virtual Machine

- Why a virtual machine?



Identifiers

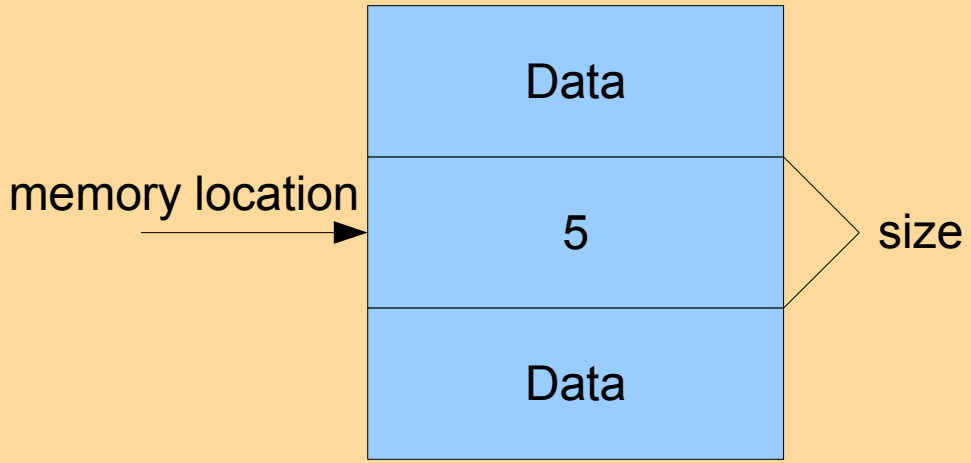
- Structure
  - Must start with letter or “_”
  - Can contain numbers
  - Examples: _local, x2, variableName
- Simple identifiers – local
  - Contains no “.”
- Compound identifiers – “global”
  - System.out.println

Reserved Words

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Variables

- A box that contains data
  - A location in memory
- The data inside the box
  - A value
- Example:
- x = x + 2



Types

- Java is strongly typed
  - Weakly typed: Hope for the best
- Apples and oranges
  - Automatic conversion or Compile error
- Types:
  - Primitive Types:
    - boolean, int, short, ...
  - Classes:
    - String, ...

Variable Declarations

- Reserves space in memory
- Makes the name (identifier) usable after this point.
- <type-name> <variable-name-or-names>
  - int numberOfStudents;
  - String name; // First, middle and last name
  - double x, y; // represents coordinates
  - boolean isFinished;
  - char firstInitial, middleInitial, lastInitial;

Type Conversion of Strings

- Integer.parseInt(“123”)
- Double.parseDouble(“3.14”)
- Double.parseDouble(“12.3e-7”)
- Same as literals
- Enum
  - Season.valueOf(“SUMMER”)

TextIO

- Class file from textbook with modifications
- Hides details of getting Input
- System.out.println(“String”)
- TextIO.put(“String”)

TextIO – printf – putf

- System.out.printf("The product of %d and %d is %d", x, y, x*y);
- Variable number of arguments
- %d – integer (decimal) number
  - %12d, minimum 12 characters
- %s – String (converted into string)
  - %10s, minimum 10 characters

TextIO – printf – putf 2

- %f – floating point
  - %12.3f – 12 characters, 3 digits after decimal point
- %e – exponential
  - %15.8e – 8 digits after the decimal point
- %g – floating point or exponential
  - %12.4g – a total of 4 digits in the answer

- " 5.345"
- " 34.453"
- " 123.875"

TextIO 2

- j = TextIO.getlnInt(); // Reads a value of type int.
- y = TextIO.getlnDouble(); // Reads a value of type double.
- a = TextIO.getlnBoolean(); // Reads a value of type boolean.
- c = TextIO.getlnChar(); // Reads a value of type char.
- w = TextIO.getlnWord(); // Reads one "word" as a value of type String.
- s = TextIO.getln(); // Reads an entire input line as a String.

TextIO – File I/O

- TextIO.writeFile("result.txt")
- TextIO.writeUserSelectedFile()
- TextIO.writeStandardOutput()
- TextIO.readFile("data.txt")
- TextIO.readUserSelectedFile()
- TextIO.readStandardInput()

Loops

- `While` loop
- `Do while` loop
- `For` loop
- We only need one of these to have a complete language
- We have several for convenience

While loops

- Two variants

```
while (<boolean-expression) {
  <statement>
}

do {
  <statements>
} while (<boolean-expression)
```

For loop examples

- Simplification

```
years = 0; // initialize the variable years
while ( years < 5 ) {   // condition for continuing loop
    interest = principal * rate;     //
    principal += interest;           // do three statements
    System.out.println(principal); //
    years++;   // update the value of the variable, years
}
```

Becomes

```
for ( years = 0; years < 5; years++ ) {
    interest = principal * rate;
    principal += interest;
    System.out.println(principal);
 }
```