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

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);`

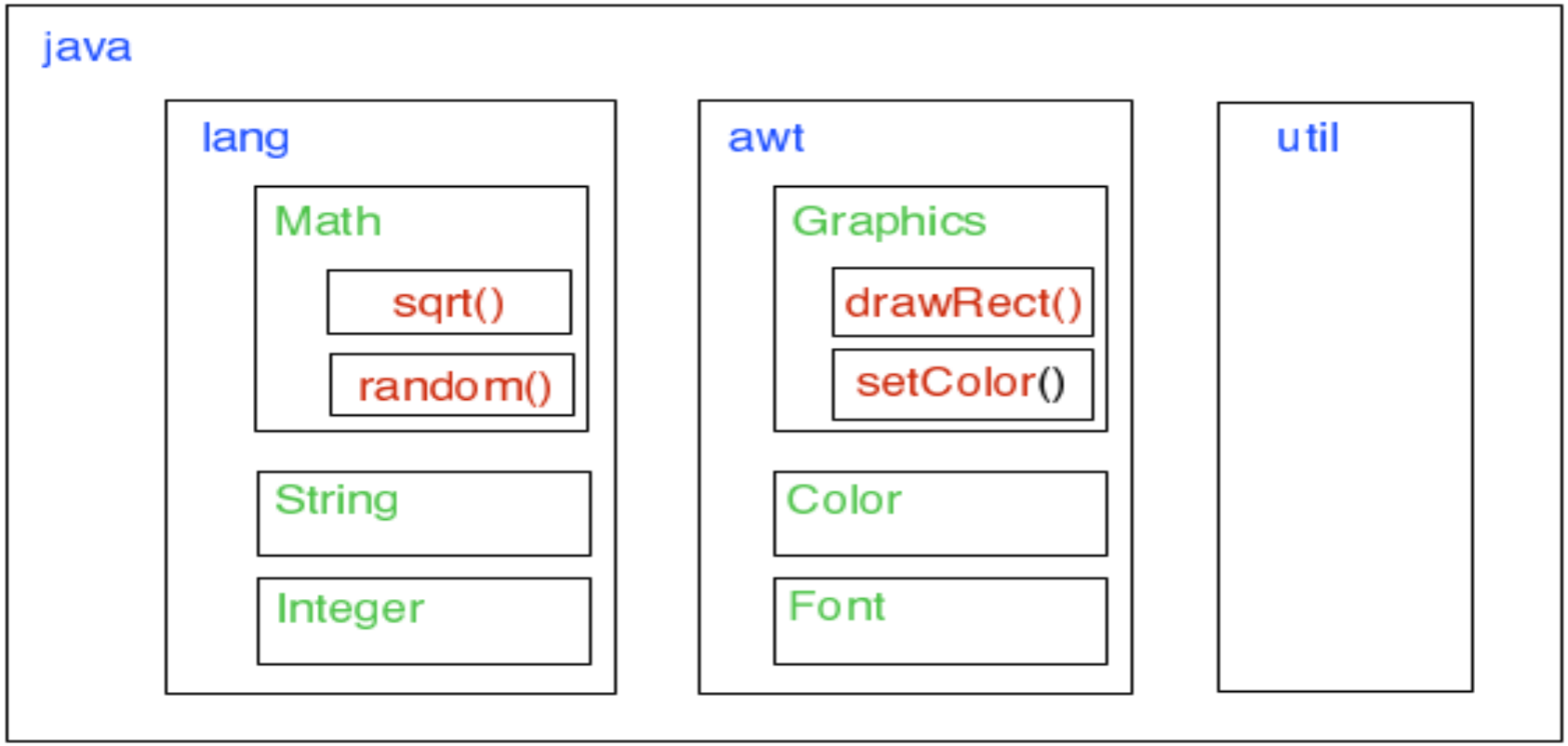
```
static String reverse(String str) {
   String copy;  // The reversed copy.
   int i;        // One of the positions in str,
                 //       from str.length() - 1 down to 0.
   copy = "";    // Start with an empty string.
   for ( i = str.length() - 1;  i >= 0;  i-- ) {

            // 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

- Lecture 1

Overview of the course

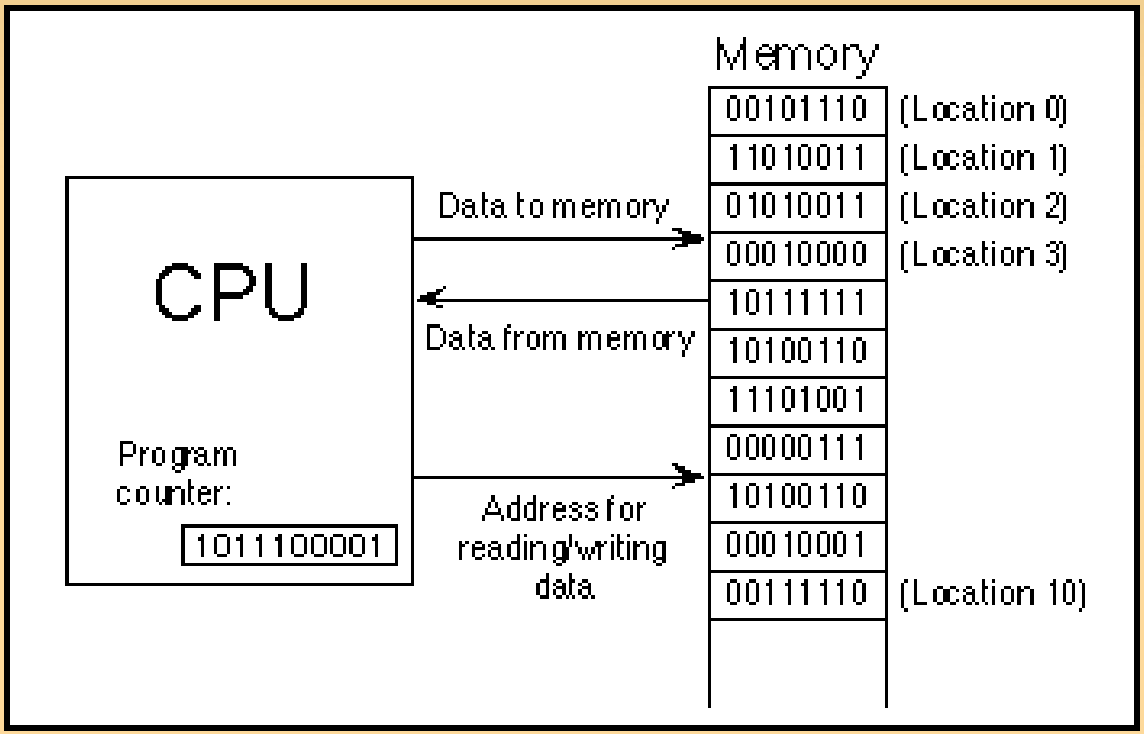
- Purpose: Learn to program
  - Basic Programming
    - Control structures, data types
    - Searching and sorting
    - Recursion
  - Knowledge of Object Oriented Programming
    - Inheritance and Polymorphism
    - Later you will have: OOP and OOA&D
- Exam: Written test

Java Virtual Machine

- Why a virtual machine
- What do we mean by “virtual”
- Explain a regular machine
- Java and Java Byte Code

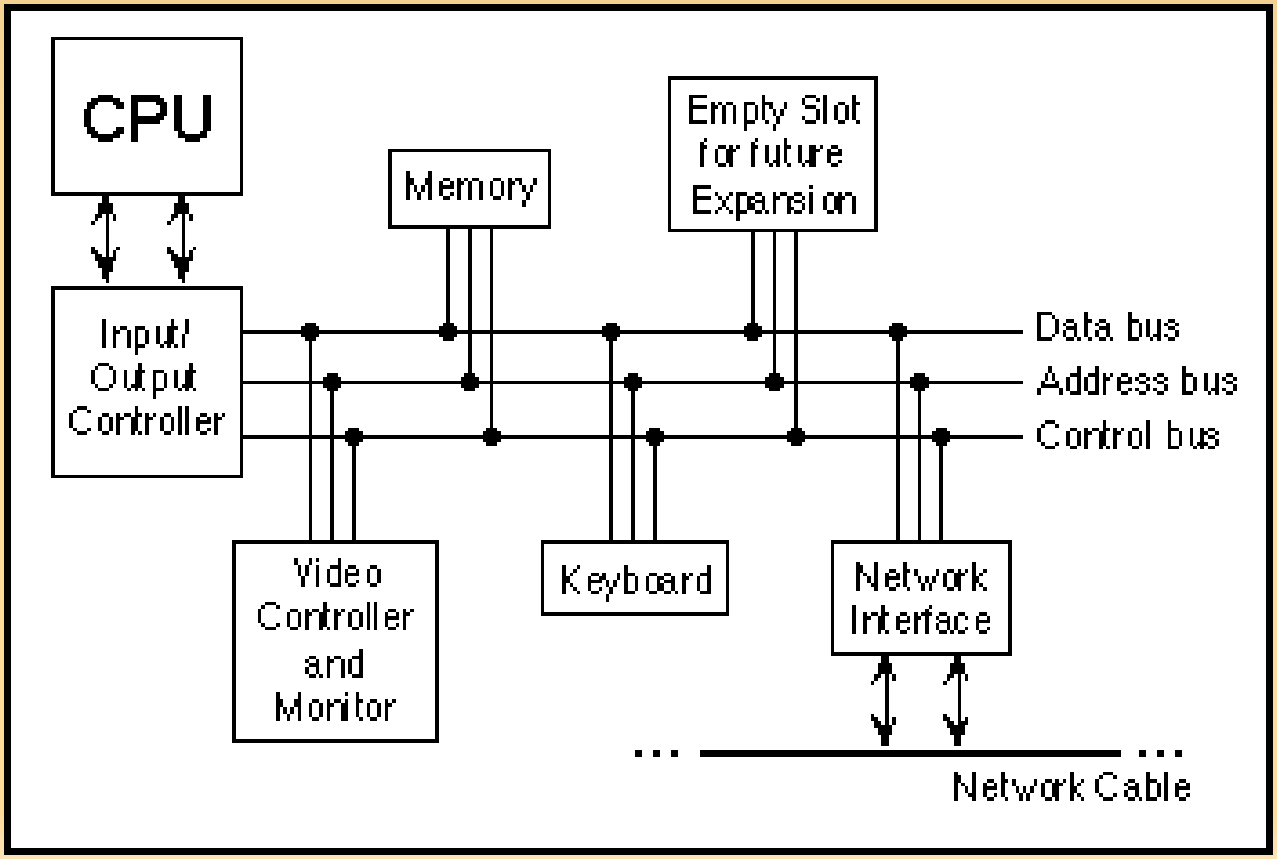
CPU

- Fetch execute cycle
- Machine language



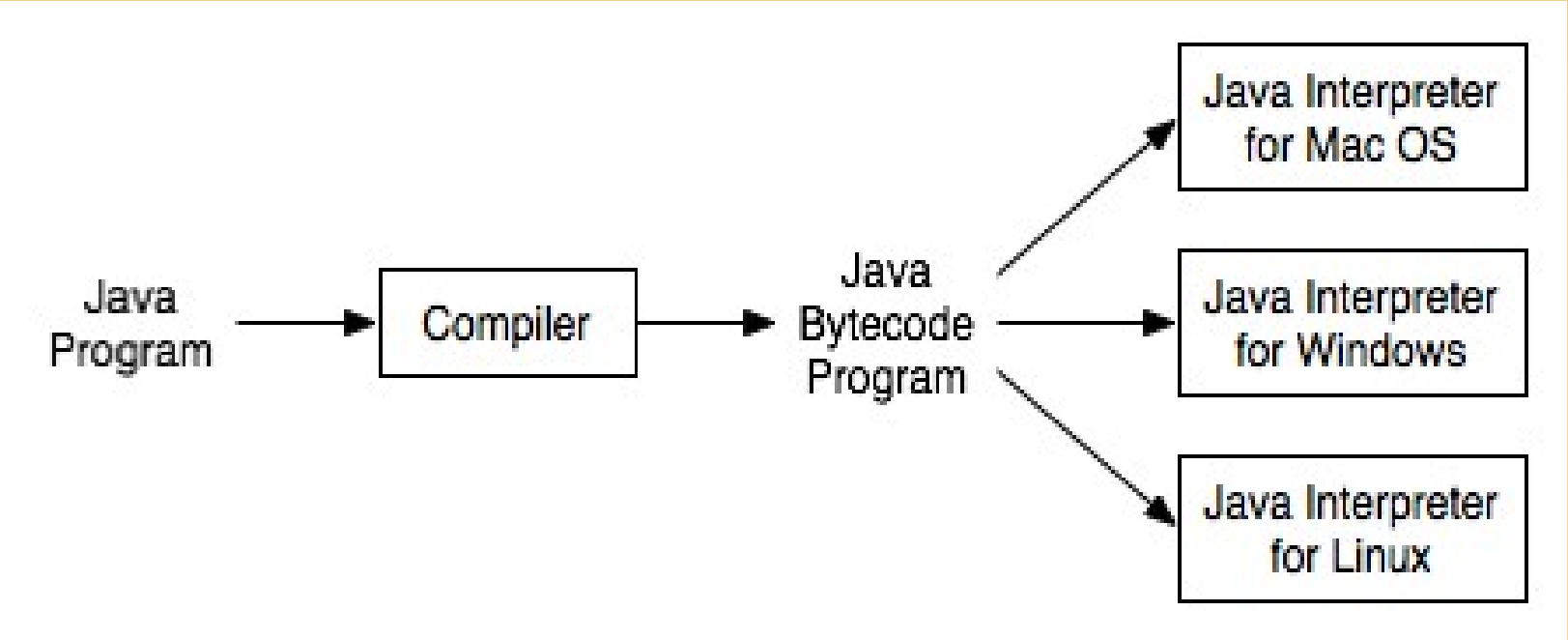
Machine Architecture

- Basic Computer Architecture
- Asynchronous events



Java Virtual Machine

- Why a virtual machine?



Compilation

```
Java Byte Code:
  0:   iconst_2
  1:   istore_1
  2:   iload_1
  3:   sipush  1000
  6:   if_icmpge       44
  9:   iconst_2
  10:  istore_2
  11:  iload_2
  12:  iload_1
  13:  if_icmpge       31
  16:  iload_1
  17:  iload_2
  18:  irem             # remainder
  19:  ifne    25
  22:  goto    38
  25:  iinc    2, 1
  28:  goto    11
  31:  getstatic       #84; //Field java/lang/System.out:Ljava/io/PrintStream;
  34:  iload_1
  35:  invokevirtual   #85; //Method java/io/PrintStream.println:(I)V
  38:  iinc    1, 1
  41:  goto    2
  44:  return
```

```
Java Code:
 for (int i = 2; i < 1000; i++) {
  for (int j = 2; j < i; j++) {
    if (i % j == 0)
      continue outer;
  }
  System.out.println (i);
 }
```

Building blocks of programs

- Data
  - Variables
  - Types
- Instructions
  - Control structures
    - organize code
  - Subroutines
    - reuse

```
Java Code:
 for (int i = 2; i < 1000; i++) {
  for (int j = 2; j < i; j++) {
    if (i % j == 0)
      continue outer;
  }
  System.out.println (i);
 }
```

History of Programming

- Structured programming
  - Divide problem into smaller problems
  - top-down approach
  - Focus on instructions, not data
- Object Oriented Programming
  - Model the problem area
  - bottom-up approach
  - Focus on data, not instructions

Object Oriented Programming

- What is an object?
  - Represents real world objects
  - Data and associated methods (functions).
    - Data hiding
    - Polymorphism
    - Classes
    - Inheritance

Data Hiding

- Ensuring
  - modularity
  - data integrity
- Enabling
  - reuse
  - local modifications

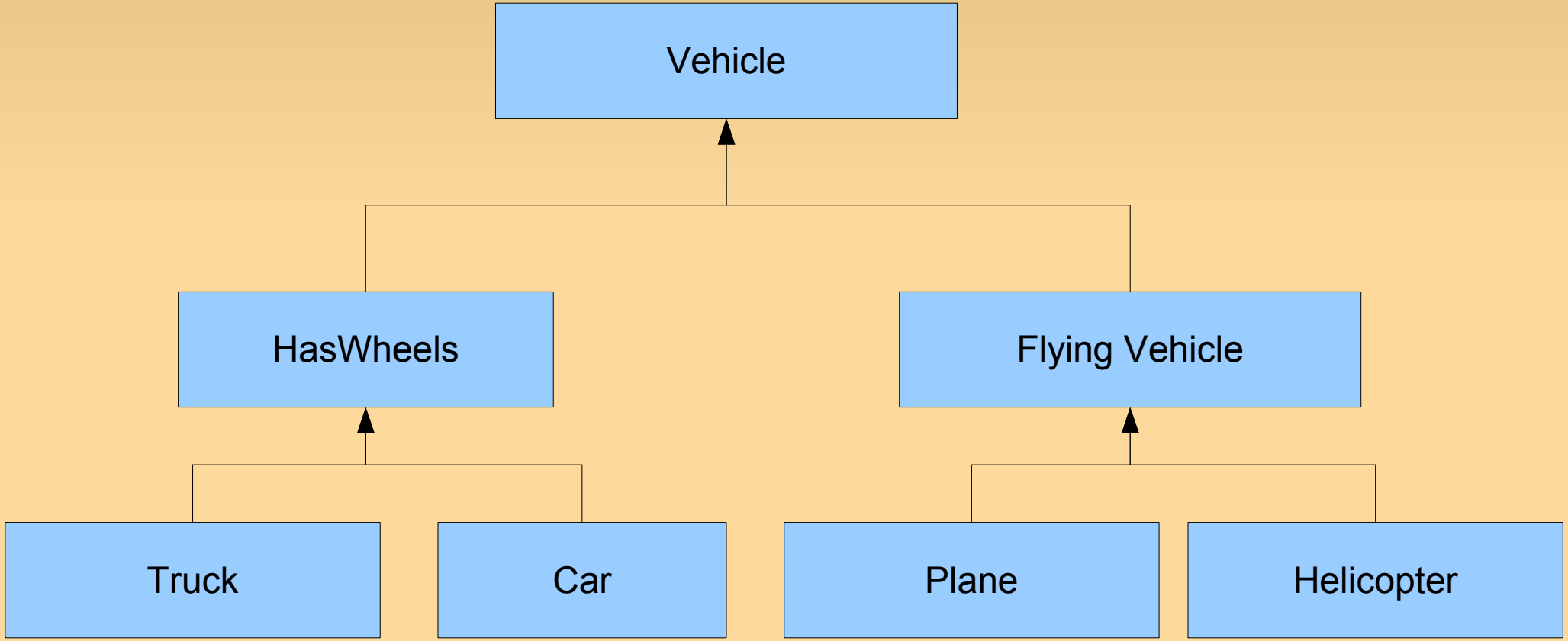
Polymorphism

- The same message send to different objects will have different effects
- Code that operates on data types that we have not defined yet

Classes

- Template
- Description of a group of objects
- Example: Vehicle

Inheritance



Command Line Interface

- Windows: Run Program (cmd)
- Linux: xterm, gterm, ...
- Mac OS: Terminal
- Javac – compiler
  - `> javac HelloWorld.java`
- Java – execution
  - `> java HelloWorld`
  - `Hello World!`

Packages

- Packages
  - `> package mypackage;`
- Compilation with packages
  - Windows
  - `> javac mypackage\HellowWorld.java`
  - Linux
  - `> javac mypackage/HellowWorld.java`

Lecture 2

Identifiers

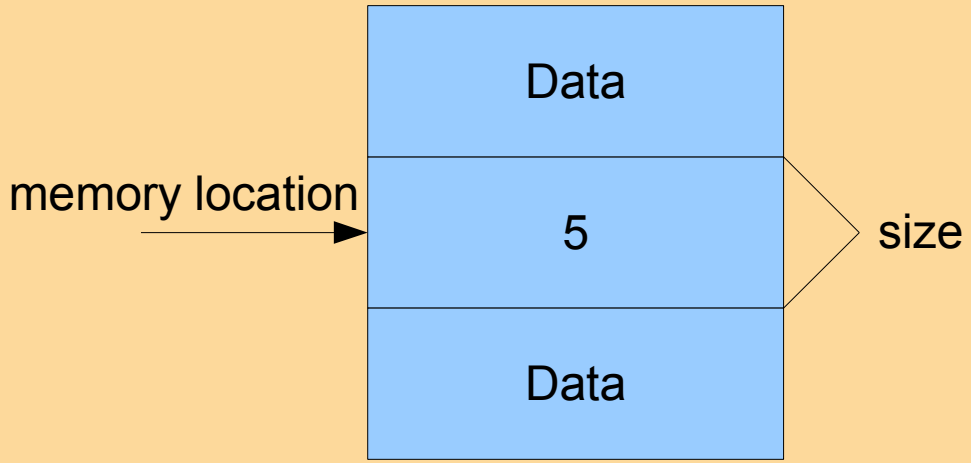
- Structure
  - Must start with letter or “_”
  - Can contain numbers
  - Examples: _local, x2, variableName
- Simple identifiers – local
  - Contains no “.”
- Compound identifiers – “global”
  - System.out.println

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, ...

Primitive Types

- Not classes
- Describes a single value
- Integer:
  - int x = 5
  - int y = 345
- Double
  - double y = 3.56 (Bemærk komma er .)

Primitive Types 2

Name	bits	Range
byte	8	-128 to 127
short	16	-32,768 to 32,767
int	32	-2,147,483,648 to 2,147,483,647
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Primitive Types 3

- float 32-bit
  - 7 significant digits
- double 64-bit
  - 15 significant digits
- boolean 1 bit of information
  - true (1) or false (0)

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;

Assignment Statements

- Putting something into the box
- <variable> = <expression>
- x = 2
- y = 4 * 5
- myVariable = x / y
- More expression later

```java
/**
 * This class implements a simple program that
 * will compute the amount of interest that is
 * earned on $17,000 invested at an interest
 * rate of 0.07 for one year.  The interest and
 * the value of the investment after one year are
 * printed to standard output.
 */

public class Interest {

   public static void main(String[] args) {

      /* Declare the variables. */

      double principal;      // The value of the investment.
      double rate;           // The annual interest rate.
      double interest;       // Interest earned in one year.

      /* Do the computations. */

      principal = 17000;
      rate = 0.07;
      interest = principal * rate;    // Compute the interest.

      principal = principal + interest;
            // Compute value of investment after one year, with interest.
            // (Note: The new value replaces the old value of principal.)

      /* Output the results. */

      System.out.print("The interest earned is $");
      System.out.println(interest);
      System.out.print("The value of the investment after one year is $");
      System.out.println(principal);

   } // end of main()

} // end of class Interest
```

Literals

- Numbers
  - int: 1200, -30
  - long: 1244L, -30L
- Floating point
  - double: 55.4
    - 12e5 = 12 * 10^5
  - float 12.3F, 24,953f

Literals 2

- String literals
  - “This is a String”
- Char literals
  - 'a', '\n', '\t', '\\'
  - '\u00E9' = é
- Boolean
  - true
  - false

Literals 3

- Hexadecimal – 0 ... 9 A ... F
  - 0x45 or 0xFF7A
  - 4 * 16 + 5
- Octal – 0 ... 7
  - 045 = 37
  - 4 * 8 + 5

Two Purposes of Classes

- Static collections of functions
  - Example: Math
    - Collection of static members
    - Math.PI
    - Math.random() // random double between 0 and 1
- Template for Objects
  - Example: String
    - String.equals()

Subroutine call statement

- <method-name>(<parameters>)
- Examples
  - System.out.println(“This is a String”)
  - Math.rand() // no parameters

String

- String firstName = “Ulrik”;
- firstName.equals(“Nyman”);
- false

String 2

- s1.equals(s2)
- s1.equalsIgnoreCase(s2)
- s1.length()
- s1.charAt(N)
- s1.substring(N,M)
- s1.toUpperCase()

Concatenation

- String + anything = String
- Examples:
  - int numberOfDays = 7;
  - “The week has “ + numberOfDays + “ days”
  -
  - numberOfDays * 2 + “ this is 14”

Math

- Math.rand()
- Math.PI // constant
- Math.sqrt(x) Square root
- Math.sin(y)
- Math.floor(double d) // returns integer

Enums

- Special type of classes
  - enum <enum-type-name> { <list-of-enum-values> }
  - enum Season { SPRING, SUMMER, FALL, WINTER }
  - Season.WINTER
  - Season vacation;
  - vacation = Season.SUMMER;
  - Season.FALL.ordinal() is 2,
  - System.out.println(vacation);

Expressions

- plus +: result = 4.0 + 3
- multiplication *: x = 3 * 4
- division /: z = 5/6 Gives an integer
  - 5.0/6 Gives a double
- modulus %: 34577 % 100 = 77
- minus -: t = 5 - 2
- unary minus -: -4

Increment and decrement

- counter = counter + 1;
- goalsScored = goalsScored + 1;
- counter = 4
- x = counter++; // x = 4 old value
- ++counter; // x = 5 new value
- goalsScored--;

Relational Operators

- A == B Is A "equal to" B?
- A != B Is A "not equal to" B?
- A < B Is A "less than" B?
- A > B Is A "greater than" B?
- A <= B Is A "less than or equal to" B?
- A >= B Is A "greater than or equal to" B?
- boolean sameSign;
- sameSign = ((x > 0) == (y > 0));

Boolean Operators

- Comparison
  - ==
  - !=
- And &&
  - true && false
- Or ||
  - true || false
- Not !
  - true == !false

Conditional Operator

- If...then...else on a single line

```
⟨boolean-expression⟩ ? ⟨expression1⟩ : ⟨expression2⟩
```

```
next = (N % 2 == 0) ? (N/2) : (3*N+1);
```

Assignment Operators

- x -= y; // same as: x = x - y;
- x *= y; // same as: x = x * y;
- x /= y; // same as: x = x / y;
- x %= y; // same as: x = x % y;
  - (for integers x and y)
- q &&= p; // same as: q = q && p;
  - (for booleans q and p)

Type Casts

- int A;
- double X;
- short B;
- A = 17;
- X = A; // OK; A is converted to a double
- B = A; // illegal; no automatic conversion
- // from int to short

Type Casts 2

- int A;
- short B;
- A = 17;
- B = (short)A; // OK; A is explicitly type cast
- // to a value of type short

Precedence Rules

- Unary operators: ++, --, !, unary - and +, type-cast
- Multiplication and division: *, /, %
- Addition and subtraction: +, -
- Relational operators: <, >, <=, >=
- Equality and inequality: ==, !=
- Boolean and: &&
- Boolean or: ||
- Conditional operator: ?:
- Assignment operators: =, +=, -=, *=, /=, %=

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 chacters, 3 digits after decimal point
- %e – exponential
  - %15.8e – 8 digits after the devimal 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()

Lecture 3

Blocks

- Grouping things together
- Simple statements and compound statement
- Statements end with ; or }

```
{
   <statements>
}
```

```
{  // This block exchanges the values of x and y
   int temp;       // A temporary variable for use in this block.
   temp = x;       // Save a copy of the value of x in temp.
   x = y;          // Copy the value of y into x.
   y = temp;       // Copy the value of temp into y.
}
```

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 loop

- Two variants

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


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

While examples

- Declare variables
- Prime loop
- Iterate

```
int number;    // The number to be printed.
number = 1;    // Start with 1.
while ( number < 6 ) { // Keep going as long as number is < 6.
    System.out.println(number);
    number = number + 1; // Go on to the next number.
}
System.out.println("Done!");
```

Do ... while loop

- Two variants

```
do
        <statement>
while (<boolean-expression>);

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

Do .. while examples

- Always executes loop body once

```
boolean wantsToContinue; // True if user wants
                         //  to play again.
do {
   Checkers.playGame();
   TextIO.put("Do you want to play again? ");
   wantsToContinue = TextIO.getlnBoolean();
} while (wantsToContinue == true);
```

For loops

- Why yet another type?
- Standard form

```
<initialization>
while (<continuation-condition>) {
   <statements>
   <update>
}
```

```
for (<init>;<continuation-cond>; <update>) {
   <statements>
}
// Also possible without block statement
```

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);
 }
```

For loop

- Standard form

```
for (<variable> = <min>;<variable> <= <max>;<variable>++ ) {
     <statements>
}
```

There is an error below

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

Off by one errors

```
for (int i = 1; i < 10; i++) {
     System.out.println(i);
}
```

Several counters

- More advanced for loops

```
for ( i=1, j=10; i <= 10; i++, j-- ) {
   TextIO.putf("%5d", i); // Output i in a 5-character wide column.
   TextIO.putf("%5d", j); // Output j in a 5-character column
   TextIO.putln();        //     and end the line.
}
```

```
    1   10
    2    9
    3    8
    4    7
    5    6
    6    5
    7    4
    8    3
    9    2
   10    1
```

Iterating over chars

- Printing out the English alphabet
- Unfortunately its more complex for Danish characters

```
// Print out the alphabet on one line of output.
char ch; // The loop control variable;
          //       one of the letters to be printed.
for ( ch = 'A'; ch <= 'Z'; ch++ )
    System.out.print(ch);
System.out.println();
```

Nested for loops

```
for ( rowNumber = 1; rowNumber <= 12; rowNumber++ ) {
   for ( N = 1; N <= 12; N++ ) {
                // print in 4-character columns
      System.out.printf( "%4d;", N * rowNumber ); // No new line!
   }
  System.out.println(); // Add a carriage return at end of the line.
}
```

```
   1;   2;   3;   4;   5;   6;   7;   8;   9;  10;  11;  12;
   2;   4;   6;   8;  10;  12;  14;  16;  18;  20;  22;  24;
   3;   6;   9;  12;  15;  18;  21;  24;  27;  30;  33;  36;
   4;   8;  12;  16;  20;  24;  28;  32;  36;  40;  44;  48;
   5;  10;  15;  20;  25;  30;  35;  40;  45;  50;  55;  60;
   6;  12;  18;  24;  30;  36;  42;  48;  54;  60;  66;  72;
   7;  14;  21;  28;  35;  42;  49;  56;  63;  70;  77;  84;
   8;  16;  24;  32;  40;  48;  56;  64;  72;  80;  88;  96;
   9;  18;  27;  36;  45;  54;  63;  72;  81;  90;  99; 108;
  10;  20;  30;  40;  50;  60;  70;  80;  90; 100; 110; 120;
  11;  22;  33;  44;  55;  66;  77;  88;  99; 110; 121; 132;
  12;  24;  36;  48;  60;  72;  84;  96; 108; 120; 132; 144;
```

Enums and for each loops

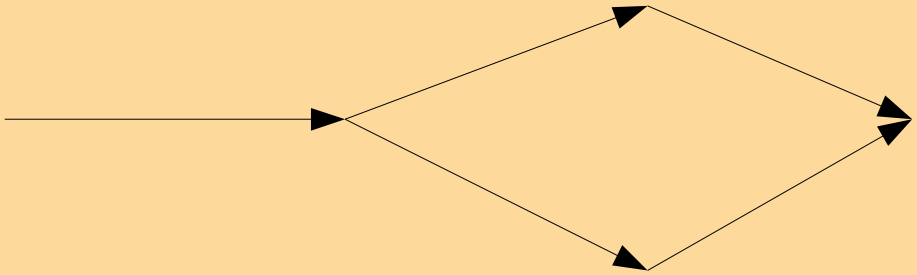
```
for (<enum-type-name> <variable-name> : <enum-type-name>.values() ) {
    <statements>
}
```

- Could have been called for each
- Printing out days and their ordinal number

```
public class DayIterator {
    // The days of the week
    enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY }

    public static void main(String[] args) {
        //
        for (Day d : Day.values() ) {
            System.out.println(d + " is day number " + d.ordinal());
        }
    }
}
```

Branches

- The `if` statement
- The `switch` statement
- Branches the computation tree



The if statement

- Basic form: Selects one of two actions

```
if (<boolean-expression> )
      <statement-1>
else
      <statement-2>
```

or

```
if (<boolean-expression>) {
      <statement-1>
} else {
      <statement-2>
}
```

The dangling else problem

- When is the second case executed

```
if ( x > 0 )
    if (y > 0)
       System.out.println("First case");
else
    System.out.println("Second case");
```

The computer reads this as

```
if ( x > 0 )
    if (y > 0)
       System.out.println("First case");
    else
        System.out.println("Second case");
```

The dangling else problem

- Solution:
  - use block statements

```
if ( x > 0 ) {
    if (y > 0)
       System.out.println("First case");
}
else
    System.out.println("Second case");
```

If...else if construction

- Used to chose between more than two things
- Example: Exactly one of the three statements are executed

```
if (<boolean-expression-1>)
      <statement-1>
else if (<boolean-expression-2>)
      <statement-2>
else
      <statement-3>
```

If...else if construction

- Can be extended indefinitely
- Last else is optinal

```
if ( boolean-expression-1 )
      statement-1
else if ( boolean-expression-2 )
      statement-2
else if ( boolean-expression-3 )
      statement-3
   .
   . // (more cases)
   .
else if ( boolean-expression-N )
      statement-N
else
      statement-(N+1)
```

The switch statement

- Most common form

```
switch ( expression ) {
    case constant-1 :
        statements-1
       break;
    case constant-2 :
        statements-2
       break;
       .
       .    // (more cases)
       .
    case constant-N :
        statements-N
       break;
    default: // optional default case
        statements-(N+1)
} // end of switch statement
```

Switch example

```
switch ( N ) {   // (Assume N is an integer variable.)
   case 1:
      System.out.println("The number is 1.");
      break;
   case 2:
   case 4:
   case 8:
      System.out.println("The number is 2, 4, or 8.");
      System.out.println("(That's a power of 2!)");
      break;
   case 3:
   case 6:
   case 9:
      System.out.println("The number is 3, 6, or 9.");
      System.out.println("(That's a multiple of 3!)");
      break;
   case 5:
      System.out.println("The number is 5.");
      break;
   default:
      System.out.print("The number is 7 or is outside");
      System.out.println(" the range 1 to 9.");
}
```

Enums and switch

```
switch ( currentSeason ) {
   case WINTER:    // ( NOT Season.WINTER ! )
      System.out.println("December, January, February");
      break;
   case SPRING:
      System.out.println("March, April, May");
      break;
   case SUMMER:
      System.out.println("June, July, August");
      break;
   case FALL:
      System.out.println("September, October, November");
      break;
}
```

Definite assignment

- A variable must be assign before it is used
- It must be assigned on all possible paths

```
String computerMove;
switch ( (int)(3*Math.random()) ) {
   case 0:
      computerMove = "Rock";
      break;
   case 1:
      computerMove = "Scissors";
      break;
   case 2:
      computerMove = "Paper";
      break;
}
System.out.println("Computer's move is " + computerMove); // ERROR!
```

Definite assignment

- Use default

```
String computerMove;
switch ( (int)(3*Math.random()) ) {
   case 0:
      computerMove = "Rock";
      break;
   case 1:
      computerMove = "Scissors";
      break;
   default:
      computerMove = "Paper";
      break;
}
System.out.println("Computer's move is " + computerMove);
```

Definite assignment

- Example 2
- No definite assignment

```
String computerMove;
int rand;
rand = (int)(3*Math.random());
if ( rand == 0 )
   computerMove = "Rock";
else if ( rand == 1 )
   computerMove = "Scissors";
else if ( rand == 2 )
   computerMove = "Paper";
```

Definite assignment

```
String computerMove;
int rand;
rand = (int)(3*Math.random());
if ( rand == 0 )
   computerMove = "Rock";
else if ( rand == 1 )
   computerMove = "Scissors";
else
   computerMove = "Paper";
```

Break and continue

- We can exit a loop prematurely

```
while (true) { // looks like it will run forever!
   TextIO.put("Enter a positive number: ");
   N = TextIO.getlnInt();
   if (N > 0)   // input is OK; jump out of loop
      break;
   TextIO.putln("Your answer must be > 0.");
}
// continue here after break
```

Labeled break

- For use in nested loops

```
boolean nothingInCommon;
nothingInCommon = true; // Assume s1 and s2 have no chars in common.
int i,j; // Variables for iterating through the chars in s1 and s2.
i = 0;
bigloop: while (i < s1.length()) {
   j = 0;
   while (j < s2.length()) {
      if (s1.charAt(i) == s2.charAt(j)) {//s1 and s2 have a common char.
          nothingInCommon = false;
          break bigloop; // break out of BOTH loops
      }
      j++; // Go on to the next char in s2.
   }
   i++; //Go on to the next char in s1.
}
```

Continue

- Jumps to the top of the loop and starts the next iteration
- Also exists in a labeled version

Algorithm development

- Pseudocode
- Stepwise refinement

Pseudocode

- English description in steps
- Can be more mathematical

```
Get the user's input
while there are more years to process:
    Compute the value after the next year
    Display the value
```

Stepwise refinement

- Get the user's input

  Ask the user for the initial investment
  Read the user’s response
  Ask the user for the interest rate
  Read the user’s response

3N+1 Problem

"Given a positive integer, N, define the '3N+1' sequence starting from N as follows: If N is an even number, then divide N by two; but if N is odd, then multiply N by 3 and add 1. Continue to generate numbers in this way until N becomes equal to 1. For example, starting from N = 3, which is odd, we multiply by 3 and add 1, giving N = 3*3+1 = 10. Then, since N is even, we divide by 2, giving N = 10/2 = 5. We continue in this way, stopping when we reach 1, giving the complete sequence: 3, 10, 5, 16, 8, 4,2, 1.

"Write a program that will read a positive integer from the user and will print out the 3N+1 sequence starting from that integer. The program should also count and print out the number of terms in the sequence."

3N+1 Problem

- Get a positive integer N from the user
- Compute, print, and count each number in the sequence;
- Output the number of terms;
-
- The second step is still very complex

3N+1 Problem

- Get a positive integer N from the user;
- while N is not 1;
  - Compute N = next term;
  - Output N;
  - Count this term;
- Output the number of terms;

3N+1 Problem

- Branch on even

```
Get a positive integer N from the user;
while N is not 1:
    if N is even:
       Compute N = N/2;
    else
       Compute N = 3 * N + 1;
    Output N;
    Count this term;
Output the number of terms;
```

3N+1 Problem

- Adding counter

```
Get a positive integer N from the user;
Let counter = 0;
while N is not 1:
    if N is even:
       Compute N = N/2;
    else
       Compute N = 3 * N + 1;
    Output N;
    Add 1 to counter;
Output the counter;
```

3N+1 Problem

- Handling incorrect input

```
Ask user to input a positive number;
Let N be the user's response;
while N is not positive:
   Print an error message;
   Read another value for N;
Let counter = 0;
while N is not 1:
    if N is even:
       Compute N = N/2;
    else
       Compute N = 3 * N + 1;
    Output N;
    Add 1 to counter;
Output the counter;
```

```
lic class ThreeN1 {
  public static void main(String[] args) {
    int N;       // for computing terms in the sequence
    int counter; // for counting the terms
    TextIO.put("Starting point for sequence: ");
    N = TextIO.getlnInt();
    while (N <= 0) {
       TextIO.put("The starting point must be positive. Please try again: ");
       N = TextIO.getlnInt();
    }
    // At this point, we know that N > 0
    counter = 0;
    while (N != 1) {
        if (N % 2 == 0)
           N = N / 2;
        else
           N = 3 * N + 1;
        TextIO.putln(N);
        counter = counter + 1;
    }
    TextIO.putln();
    TextIO.put("There were ");
    TextIO.put(counter);
    TextIO.putln(" terms in the sequence.");
 }  // end of main()
// end of class ThreeN1
```

Debugging

- Debugging statements
- System.out.println(“x=“ + x + “ before the loop”);

Lecture 4

Exceptions

- Last time: 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." );
}
```

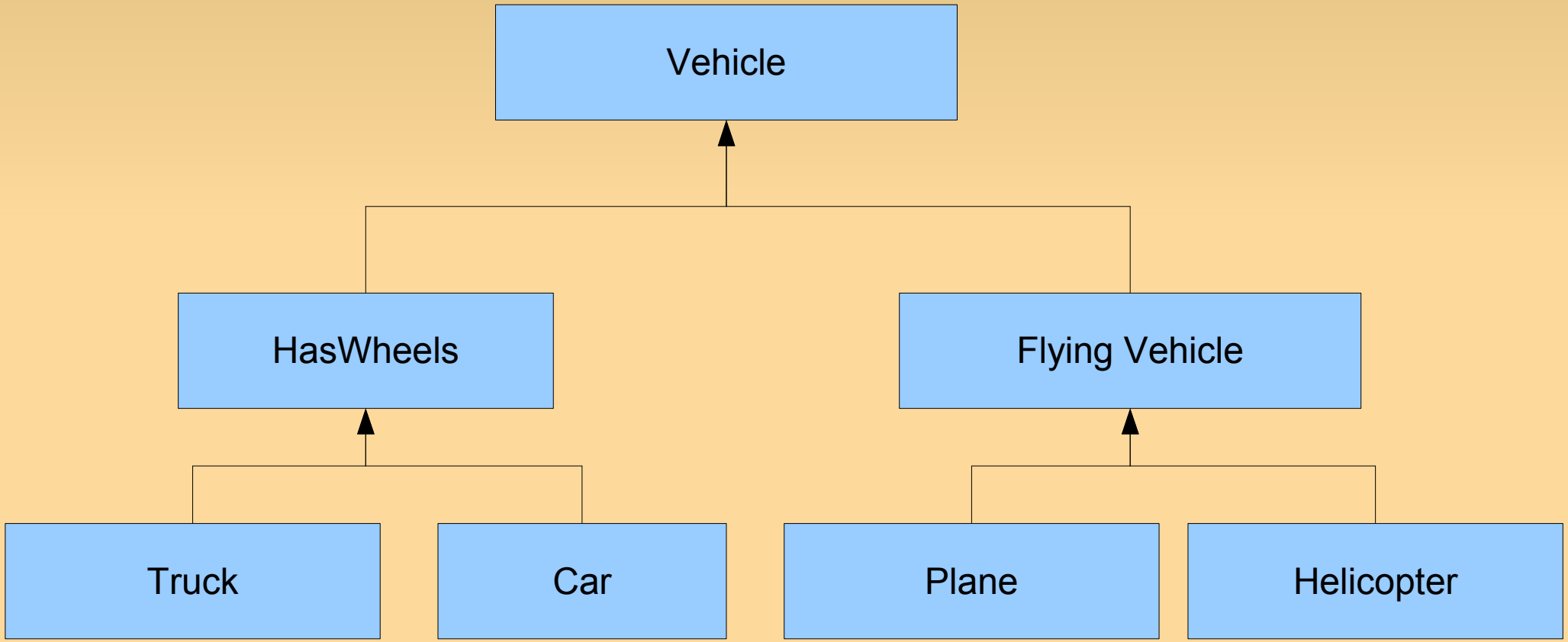
Example 2

```
Day  weekday;  // User's response as a value of type Day.
while ( true ) {
   String response;  // User's response as a String.
   TextIO.put("Please enter a day of the week: ");
   response = TextIO.getln();
   response = response.toUpperCase();
   try {
      weekday = Day.valueOf(response);
      break;
   }
   catch ( IllegalArgumentException e ) {
      TextIO.putln( response + " is not the name of a day of the week." );
   }
}
```

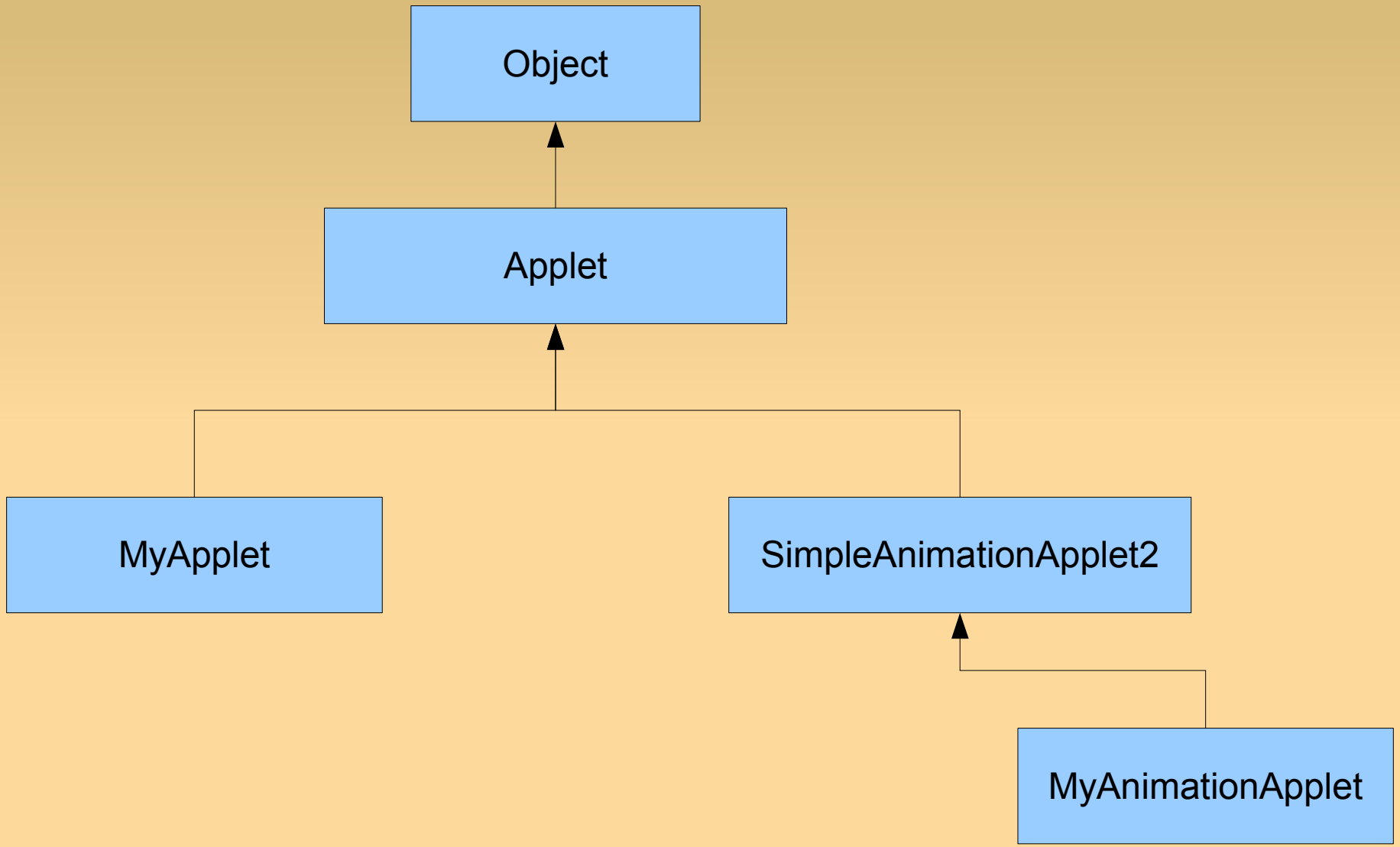
Exceptions in TextIO

- When reading user input TextIO handles error itself
- When reading from a file, this is not possible.
- Thus is throws an error that you have to catch.

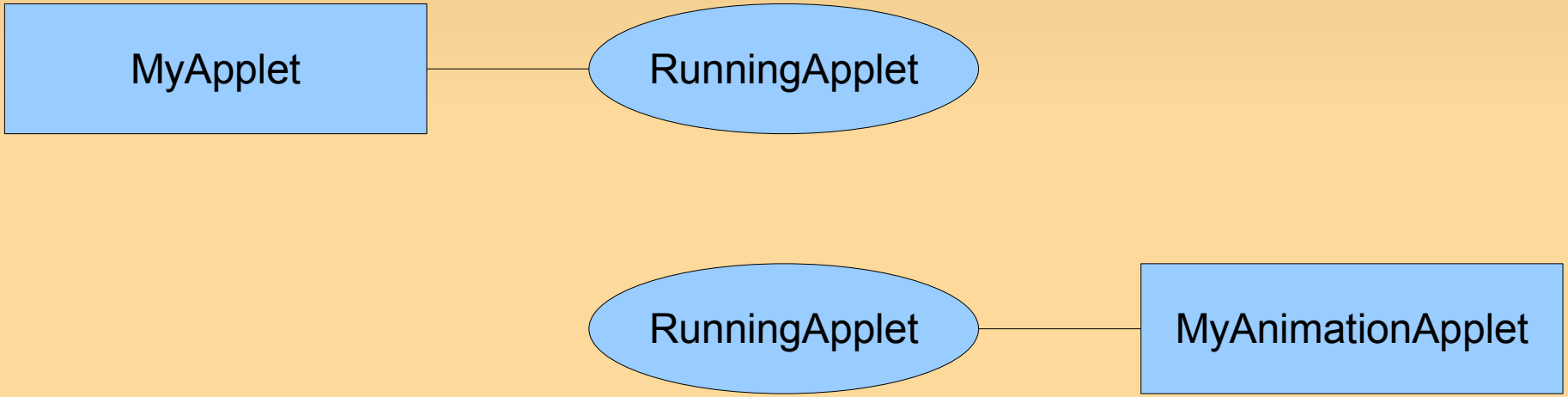
Overview


Vehicle
HasWheels
Flying Vehicle
Truck
Car
Plane
Helicopter

Classes


Object
Applet
MyApplet
SimpleAnimationApplet2
MyAnimationApplet

Static vs. non-static

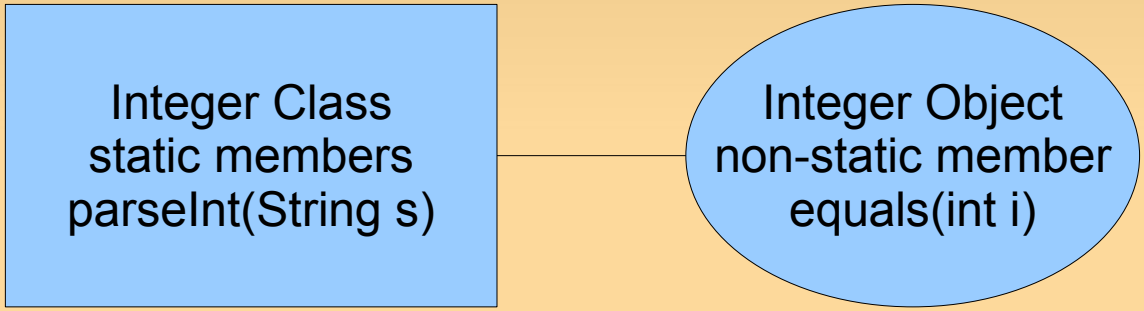


Math vs. String

- Static memebers
  - Math.rand()
  - Integer.parseInt(“45”)
- Non-static members
  - s1 = “Dette bliver et object”
  - s1.equals(“hej”)

Example: Integer

- Class vs. Object



GUI programming

- GUI = Graphical User Interface
- Applets
  - Making an applet class
  - When running the applet class an object is created
- Regular programs – as in previous lectures
  - public static void main(String[] args) {...
- Applets
  - public void paint(Graphics g) {...

Syntax

- import – packages
- extends

```
import java.awt.*;
import java.applet.*;

public class ⟨name-of-applet⟩ extends Applet {

    public void paint(Graphics g) {
       ⟨statements⟩
    }

}
```

Graphics

- java.awt vs. javax.swing
- Applet vs. Japplet
- Methods on Graphics obejct
  - g.setColor(c)
    - c is of type enum Color: ex Color.RED, Color.BLUE
  - g.drawRect(x,y,w,h)
  - g.fillRect(x,y,w,h)
  - Draws rectangles

```
import java.awt.*;
import java.applet.Applet;

public class StaticRects extends Applet {

   public void paint(Graphics g) {

         // Draw a set of nested black rectangles on a red background.
         // Each nested rectangle is separated by 15 pixels on
         // all sides from the rectangle that encloses it.

      int inset;    // Gap between borders of applet
                    //         and one of the rectangles.

      int rectWidth, rectHeight;   // The size of one of the rectangles.

      g.setColor(Color.red);
      g.fillRect(0,0,300,160);  // Fill the entire applet with red.

      g.setColor(Color.black);  // Draw the rectangles in black.

      inset = 0;

      rectWidth = 299;    // Set size of first rect to size of applet.
      rectHeight = 159;

      while (rectWidth >= 0 && rectHeight >= 0) {
         g.drawRect(inset, inset, rectWidth, rectHeight);
         inset += 15;        // Rects are 15 pixels apart.
         rectWidth -= 30;    // Width decreases by 15 pixels
                             //              on left and 15 on right.
         rectHeight -= 30;   // Height decreases by 15 pixels
                             //              on top and 15 on bottom.
      }

   }  // end paint()

}  // end class StaticRects
```

Animation

- Extend SimpleAnimationApplet2
- implement drawFrame() method
- use this.getFrameNumer() in some way

Black Boxes

- Why?
  - Hiding details and complexity
- Well defined interface
- You should not know how it is implemented
  - implementation can e changed later
- The black box should not know how it will be used later
  - it can be used in many unexpected ways

Contract

- Interface and description can be seen as a contract
- Read the description
  - fillRect(x,y,h,w)
  - Not fillRect(x1,y1,x2,y2)

Static subroutines and variables

- Subroutine definition
- modifiers – static and public, private, protected
- return-type – void or typename
- parameter-list – next slide

```
⟨modifiers⟩ ⟨return-type⟩ ⟨subroutine-name⟩ ( ⟨parameter-list⟩ ) {
    ⟨statements⟩
}
```

```
public static void main(String[] args) { ... }
```

Calling subroutines

- Inside the class
  - playGame()
- Outside the class
  - Poker.playGame()
  - Integer.parseInt(“33”)

```
⟨subroutine-name⟩(⟨parameters⟩);
```

```
⟨class-name⟩.⟨subroutine-name⟩(⟨parameters⟩);
```

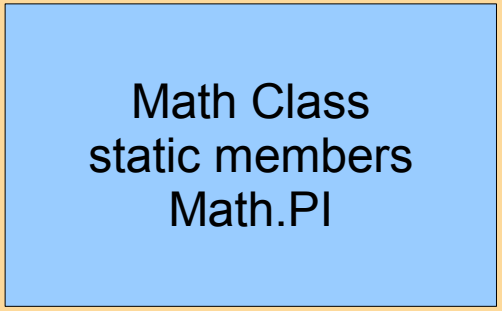
Subroutines in programs

- Split problem into smaller parts
- Use the same subroutine in several places
- Simple main loop

```
public class GuessingGame {

   public static void main(String[] args) {
      TextIO.putln("Let's play a game.  I'll pick a number between");
      TextIO.putln("1 and 100, and you try to guess it.");
      boolean playAgain;
      do {
         playGame();  // call subroutine to play one game
         TextIO.put("Would you like to play again? ");
         playAgain = TextIO.getlnBoolean();
      } while (playAgain);
      TextIO.putln("Thanks for playing.  Goodbye.");
   } // end of main()
```

Member variables

- We only look at static member variables
  - for now
- These belong to the class not the individual object
- Example PI, which is also final



Static member variables

- Static does not mean final
- Local variables in subroutines
- Global variables in classes
  - can be public or private

```
static String usersName;
public static int numberOfPlayers;
private static double velocity, time;
```

Parameters

- You have called lost of methods with parameters
- These are called actual parameters
- Formal parameters
  - The one you write when you define a subroutine, that others can call
- Actual parameters are substituted for the formal ones

Parameters

- Formal parameters
  - only declared once

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

- Actual parameters
  - as many times as you call the method
  - print3NSequence(17);

```
do {
   TextIO.putln("Enter a starting value;")
   TextIO.put("To end the program, enter 0: ");
   K = TextIO.getInt();  // Get starting value from user.
   if (K > 0)   // Print sequence, but only if K is > 0.
      print3NSequence(K);
} while (K > 0);   // Continue only if K > 0.
```

Subroutine example

```
static void doTask(int N, double x, boolean test) {
    // statements to perform the task go here
}
```

```
doTask(17, Math.sqrt(z+1), z >= 10);
```

```
{
  int N;       // Allocate memory locations for the formal parameters.
  double x;
  boolean test;
  N = 17;                 // Assign 17 to the first formal parameter, N.
  x = Math.sqrt(z+1);     // Compute Math.sqrt(z+1), and assign it to
                          //    the second formal parameter, x.
  test = (z >= 10);       // Evaluate "z >= 10" and assign the resulting
                          //     true/false value to the third formal
                          //     parameter, test.
   // statements to perform the task go here
}
```

Overloading

- Many methods with the same name
- TextIO example
  - putln(int)
  - putln(String)
  - putln(boolean)
- No overloading on return-type

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.)
   .
```