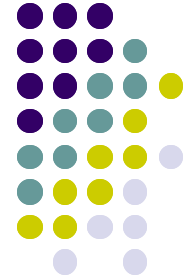
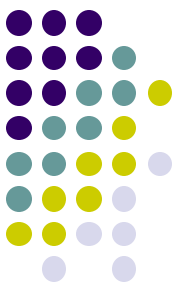


Java introduction

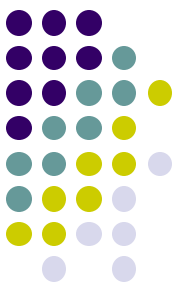


Ionut
Spalatelu



Outline

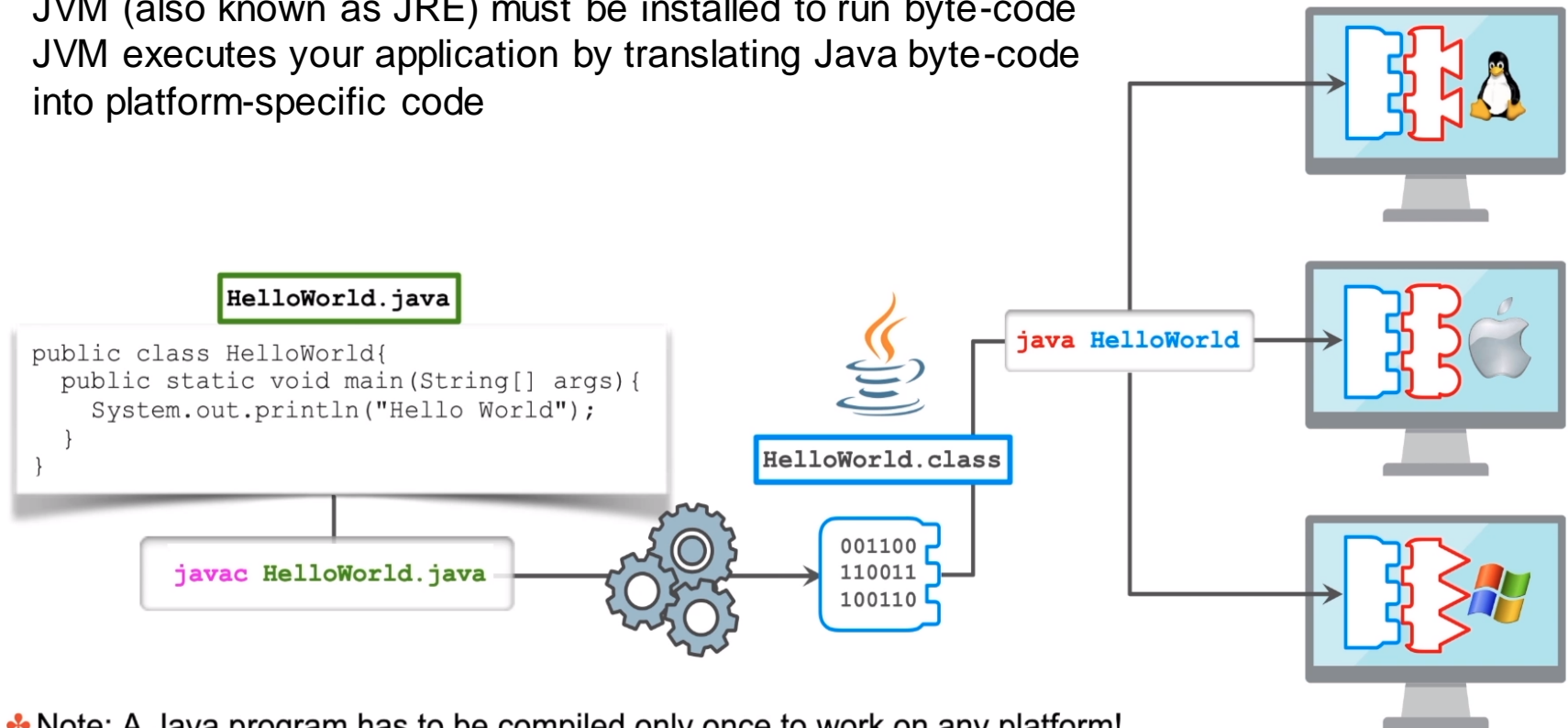
- How Java works?
- Classes
- Objects
- Java APIs
- Java naming conventions
- Syntax rules
- Imports
- Access modifiers
- Comments and docs
- Keywords
- Variables
- Data types
- Operators
- Control flow statements



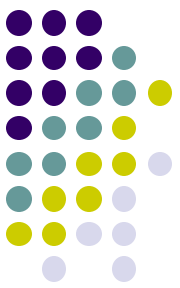
How Java works?

Java is a platform independent language

- Java source code is written as plain text .java files
- Source code is compiled into byte-code, .class files for JVM (Java Virtual Machine)
- JVM (also known as JRE) must be installed to run byte-code
- JVM executes your application by translating Java byte-code into platform-specific code



❖ Note: A Java program has to be compiled only once to work on any platform!



Classes

Class and Object are two key object oriented concepts

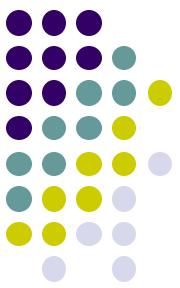
Java code is structured in **classes**.

- Class represents a type of thing or concept, such as Dog, Cat, Computer, Person etc
- Each class defines what kind of information (**attributes**) it can store:
 - A Dog could have a name, color, weight
 - A Person could have a name, age, height, and so on.
- Each class defines what kind of behaviors (**operations**) containing program logic (**algorithms**) it is capable of:
 - A Dog could bark and bring a ball
 - A Cat could meow but is not likely to play fetch.

```
class Person {  
    void play() {  
        Dog dog = new Dog();  
        dog.name = "Rex";  
        Ball ball = new Ball();  
        dog.fetch(ball);  
    }  
}
```

```
class Dog {  
    String name;  
    fetch(Ball ball) {  
        ball.find();  
        ball.chew();  
    }  
}
```

Objects



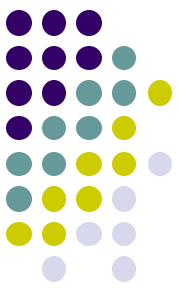
An Object is a specific **instance** (entity) of a Class

- Each object would be capable of having **specific values** for each attribute defined by a class that represents its type. For example:
 - A dog could be called Terry, and be brown and small;
 - Another could be called Rex, and be orange and big
- To operate an object, you can **reference** it using a variable of the relevant type:
 - At runtime, objects invoke operations upon each other to execute logic

```
class Person {  
    void play() {  
        Dog dog = new Dog();  
        dog.name = "Rex";  
        Ball ball = new Ball();  
        dog.fetch(ball);  
    }  
}
```

```
class Dog {  
    String name;  
    fetch(Ball ball) {  
        ball.find();  
        ball.chew();  
    }  
}
```

Java APIs

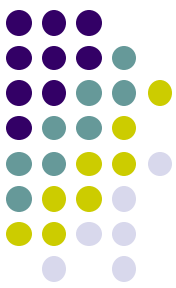


Java Development Kit (JDK) provides hundreds of classes for various programming purposes:

- To represent basic data types: Strings, LocalDateTime, BigDecimal etc
- To manipulate collections, for example: ArrayList, HashSet, HashMap etc
- To handle generic behaviors and perform system actions: System, Object, Class etc
- To perform input/output (I/O) operations, for example: FileInputStream, FileOutputStream and so on.
- Many other API classes are used to access databases, manage concurrency, enable network communications, execute scripts, security, build GUIs etc

~ API stands for Application Programming Interface and describes a collection of classes that are designed to serve a common purpose.

~ All Java APIs are documented for each version of the language. For Java 11 docs can be found at: <https://docs.oracle.com/en/java/javase/11/docs/api/>



Java keywords

Since Java 1.0

No longer in use

Since Java 1.2

Since Java 1.4

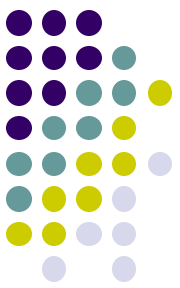
Since Java 5.0

Since Java 9.0

Reserved for
literals

special identifier
added in Java 10

if	implements	boolean	assert
else	extends	try	enum
continue	interface	catch	module
break	class	finally	requires
for	static	throw	transitive
do	final	throws	exports to
while	return	new	uses
switch	transient	this	provides
case	void	super	with
default	byte	instanceof	opens to
private	short	native	
protected	int	synchronized	true
public	long	volatile	false
import	char	goto	null
package	float	const	
abstract	double	strictfp	var



Java naming conventions

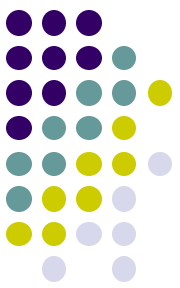
- Java is case-sensitive; Dog is not the same as dog
- Package name usually should be similar to your company name, plus the naming system adopted internally
- Class name should be a noun, in mixed case with the first letter of each word capitalized
- Variable name should be in mixed case starting with a lowercase letter; further words starting with capital letters
- Constants are typically written in uppercase with underscore symbols between words
- Method name should be a verb, in mixed case starting with a lowercase letter; further words start with capital letters.
- Syntactically, numeric characters (0-9), _ and \$ symbols are allowed in the names too but their use is discouraged and are not allowed at the beginning

```
package:    com.oracle.demos.animals  
class:      ShepherdDog  
variable:  shepherdDog  
constant:  MIN_SIZE  
method:    giveMePaw
```



```
package:    animals  
class:      Shepherd Dog  
variable:  _price  
constant:  minSize  
method:    xyz
```

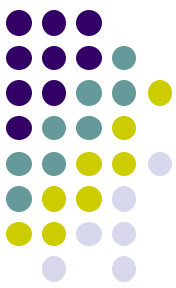




Java syntax rules

- All Java statements must be terminated with ";" symbol
- Code blocks must be enclosed with "{" and "}" symbols
- Indentations and spaces help readability, but they are not relevant to compiler

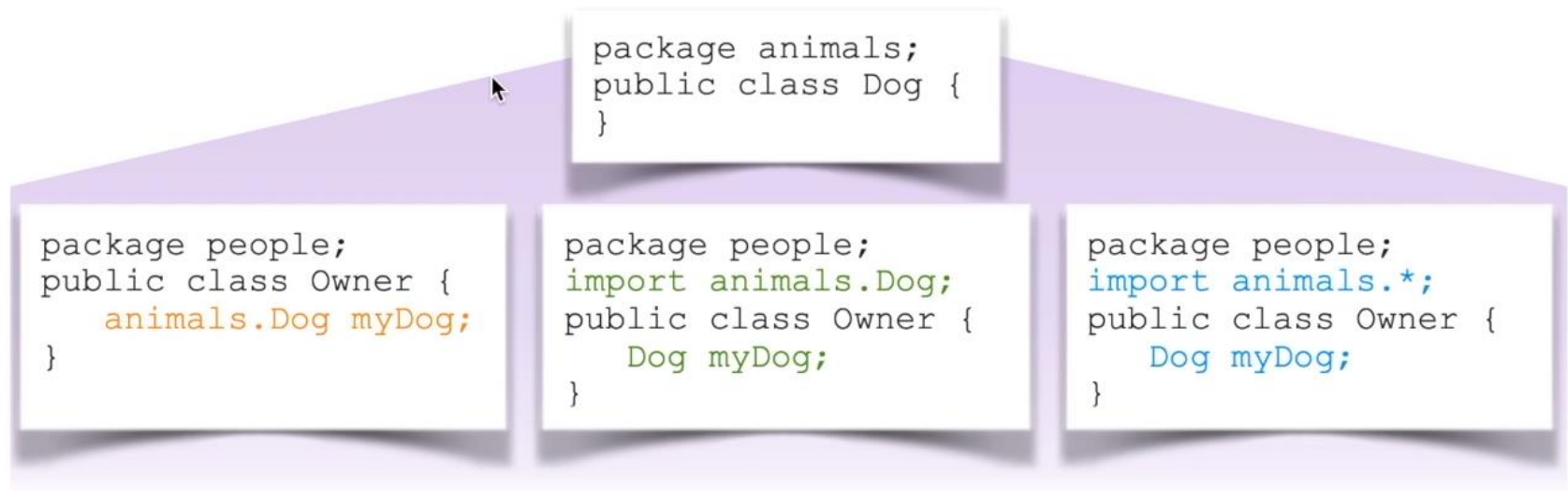
```
package com.oracle.demos.animals;  
class Dog {  
    void fetch() {  
        while (ball == null) {  
            keepLooking();  
        }  
    }  
    void makeNoise() {  
        if (ball != null) {  
            dropBall();  
        } else {  
            bark();  
        }  
    }  
}
```



Access classes across packages

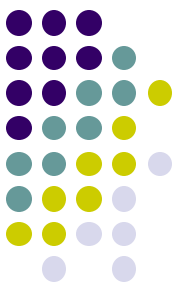
To access a class in another package:

- Prefix the class name with the package name
- Use the import statement to import specific classes of the entire package content
 - Import of the java.lang.* package is implicitly assumed



Note:

Imports are not present in the compiled code. An import statement has no effect at runtime efficiency of the class. It is a simple convenience to avoid lengthy class names with packages.



Use access modifiers

Access modifiers describe visibility of classes, variables , and methods

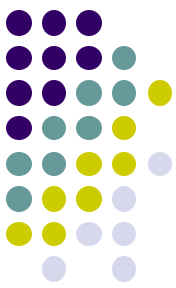
- **public** – Visible to any other class
- **protected** – Visible to classes that are in the same package or subclasses
- **<default>** – Visible only to classes in the same package
- **private** – Visible only within the same class

```
package <package name>;
import <package name>.<class name>;
import <package name>.*;
<access modifier> class <ClassName> {
    <access modifier> <variable definition>
    <access modifier> <method definition>
}
```

```
package b;
import a.*;
public class Y extends X {
    public void doThings() {
        X x = new X();
        x.y1;
        x.y2;
        x.y3;
        x.y4;
    }
}
```

```
package a;
public class X {
    public Y y1;
    protected Y y2;
    Y y3;
    private Y y4;
}
```





Comments and documentation

- **Code comments** can be placed anywhere in the source code
- **Documentation comments**
 - may contain HTML markups
 - may contain descriptive tags prefixed with @ sign
 - are used by the `javadoc` tool to generate documentation

Package demos

Class Whatever

java.lang.Object
demos.Whatever

public class Whatever
extends Object

The Whatever class represents an example of documentation comment

Version:
1.0

Author:
oracle

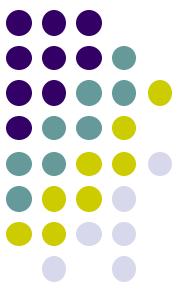
javadoc

```
// single-line comment

/*
    multi-line comment
*/

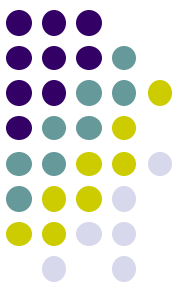
/**
 * The {@code Whatever} class
 * represents an example of
 * documentation comment
 * @version 1.0
 * @author oracle
 */
```

```
javadoc -d <documentation path>
        -sourcepath <source code path>
        -subpackages <name of the root package>
```



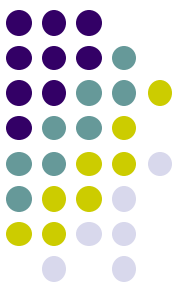
Variables

- A way of storing information inside the computer
- As its name suggests, it's content can be changed



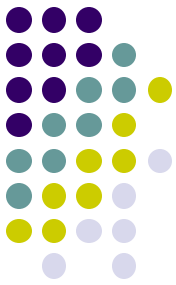
Variables

- A way of storing information inside the computer
- As its name suggests, it's content can be changed
- So, to define a variable we need to tell computer what type of information we need to store in it, and give it a name



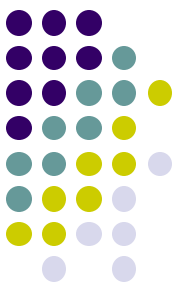
Variables

- A way of storing information inside the computer
- As its name suggests, it's content can be changed
- So, to define a variable we need to tell computer what type of information we need to store in it, and give it a name
- There are lots of different types of data that can be used to define our variables, also known as **data types**



Variables

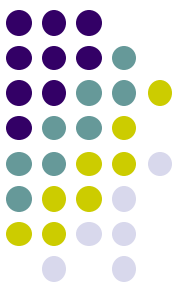
Identifier Type	Rules for Naming	Examples
Packages	<p>The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981.</p> <p>Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.</p>	<pre>com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese</pre>
Classes	<p>Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).</p>	<pre>class Raster; class ImageSprite;</pre>
Interfaces	<p>Interface names should be capitalized like class names.</p>	<pre>interface RasterDelegate; interface Storing;</pre>
Methods	<p>Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.</p>	<pre>run(); runFast(); getBackground();</pre>
Variables	<p>Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign \$ characters, even though both are allowed.</p> <p>Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.</p>	<pre>int i; char c; float myWidth;</pre>
Constants	<p>The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)</p>	<pre>static final int MIN_WIDTH = 4; static final int MAX_WIDTH = 999; static final int GET_THE_CPU = 1;</pre>



Data types

Terminology

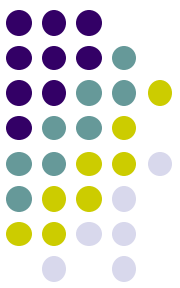
- **Data type** = a set of values (definition domain) and a set of operations defined on them
- 8 **primitive** (**built-in**) data types in Java, mostly different types of **numbers**
- OOP is centered around the idea of creating **our own data types** out of existing ones (we'll see later)



Primitives

Integer numbers

- **byte:** range -2^7 and 2^7-1 (8 bits = 1 byte)
- **short:** range -2^{15} and $2^{15}-1$ (16 bits = 2 bytes)
- **char:** range 0 to 65535 (16 bits = 2 bytes)
- **int:** range -2^{31} and $2^{31}-1$ (32 bits = 4 bytes)
- **long:** range -2^{63} and $2^{63}-1$ (64 bits = 8 bytes)

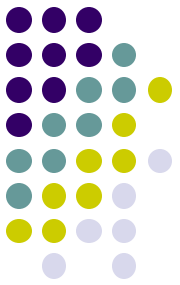


Primitives

Real numbers

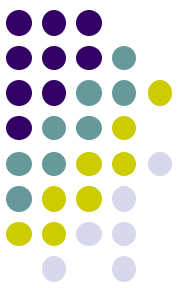
- **float:** range (32 bits = 4 bytes)
- **double:** range (64 bits = 8 bytes)
- Never use double or float for money, instead **BigDecimal**

Primitives




Booleans


- **boolean:** only values **true** and **false**

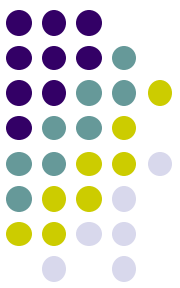


Primitives

Examples:

```
int a = 0b101010; // binary
short b = 052; // octal
byte c = 42; // decimal
long d = 0x2A; // hex
float e = 1.99E2F;
double f = 1.99;
long g = 5, h = c;
float i = g;
char j = 'A';
char k = '\u0041', l = '\101';
int s;
s = 77; 
```

```
byte a;
byte b = a;
byte c = 128;
int d = 42L;
float e = 1.2;
char f = "a";
char g = 'AB';
boolean h = "true";
boolean i = 'false';
boolean j = 0;
boolean k = False; 
```



Primitives

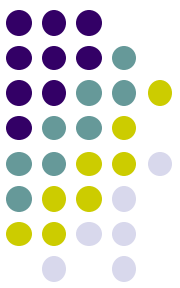
Terminology

```
int a, b, c;  
a = 5;  
b = 6;  
c = a + b;  
int d = 0;
```

The first statement is declaring 3 variables of type **int** in the same time.

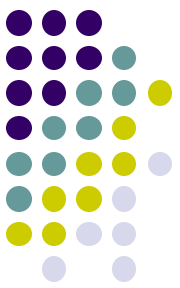
The next 2 assignment statements change the values of the variables using the **literals** 5 and 6.

The last 2 statements assign **c** the value of the expression `a + b`, and define and initialize in the same time variable **d**



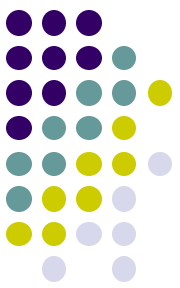
Unicode

- Is an international encoding standard for use with different languages and scripts by which each letter, digit, or symbol is assigned a **unique** numeric value that applies across different platforms and programs.



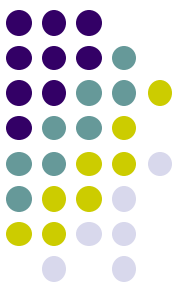
Unicode

- Is an international encoding standard for use with different languages and scripts by which each letter, digit, or symbol is assigned a unique numeric value that applies across different platforms and programmes.
- in the Latin alphabet, we've got the letters A through Z, meaning only 26x2 characters are needed in total to represent the entire Latin alphabet. But other languages need a lot more characters;



Unicode

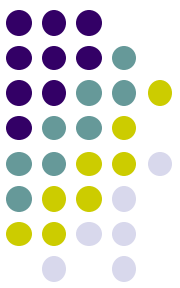
- Is an international encoding standard for use with different languages and scripts by which each letter, digit, or symbol is assigned a unique numeric value that applies across different platforms and programmes.
- in the latin alphabet, we've got the letters A through Z, meaning only 26x2 characters are needed in total to represent the entire latin alphabet. But other languages need a lot more characters;
- Unicode was developed to allow us to represent these languages in a common way on computers and it can represent any one of 65,535 different types of characters.



Escape sequences

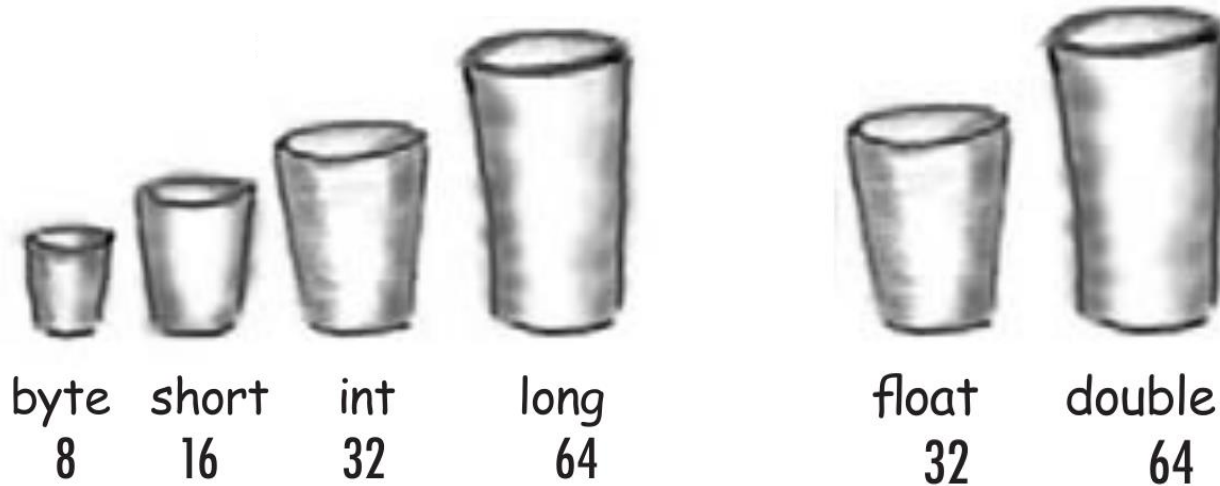
Escape Sequences

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

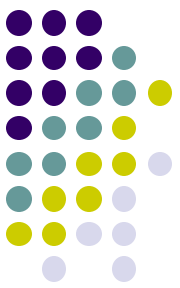


Casting in Java

- Converting a number from a type to another type
- Smaller types are automatically casted (promoted) to bigger types.
byte -> short -> char -> int -> long -> float -> double
- A bigger type value cannot be assigned to a smaller type variable without explicit casting

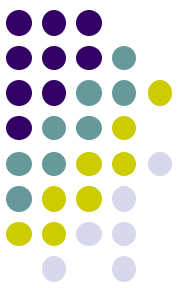


Note: parsing is not casting



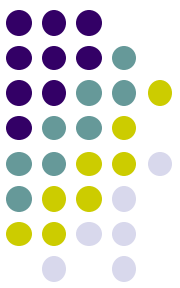
Overflow and underflow

- Overflow: occurs when you put in a variable a value larger than the maximum value of the variable's data type
- Underflow: occurs when you put in a variable a value smaller than the minimum value of the variable's data type



Operators

Type	
Arithmetic	<code>+, -, /, *, %</code> <code>--, ++</code>
Relational	<code><, >, >=, <=, ==</code>
Bitwise	<code>&, , ^, ~, <<, >>, >></code>
Logical	<code>&&, , !</code>
Assignment	<code>=, +=, -=, *=, /=, %=</code>
Misc	<code>ternary (? :)</code> <code>instanceof</code>



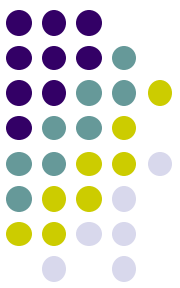
Operators

```
int a = 1;    // assignment (a is 1)
int b = a+4;  // addition (b is 5)
int c = b-2;  // subtraction (c is 3)
int d = c*b;  // multiplication (d is 15)
int e = d/c;  // division (e is 5)
int f = d%6;  // modulus (f is 3)
```

```
int a = 1, b = 3;
a += b; // equivalent of a=a+b (a is 4)
a -= 2; // equivalent of a=a-2 (a is 2)
a *= b; // equivalent of a=a*b (a is 6)
a /= 2; // equivalent of a=a/2 (a is 3)
a %= a; // equivalent of a=a%a (a is 0)
```

```
int a = 2, b = 3;
int c = b-a*b;    // (c is -3)
int d = (b-a)*b;  // (c is 3)
```

```
int a = 1, b = 0;
a++;    // increment (a is 2)
++a;    // increment (a is 3)
a--;    // decrement (a is 2)
--a;    // decrement (a is 1)
b = a++; // increment postfix (b is 1, a is 2)
b = ++a; // increment prefix  (b is 3, a is 3)
b = a--; // decrement postfix (b is 3, a is 2)
b = --a; // decrement prefix  (b is 1, a is 1)
```



Short-circuit operators

Short-circuit operators enable you to not evaluate the right-hand side of the AND and OR expressions, when the overall result can be deduced from the left-hand side value

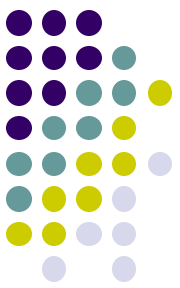
`&& ||` (short-circuit evaluation)

`& | ^` (full evaluation)

```
true && evaluated
false && not evaluated
false & evaluated
false || evaluated
true || not evaluated
true | evaluated
true ^ evaluated
false ^ evaluated
```

```
int a = 3, b = 2;
boolean c = false;
c = (a > b && ++b == 3); // c is true, b is 3
c = (a > b && ++b == 3); // c is false, b is 3
c = (a > b || ++b == 3); // c is false, b is 4
c = (a < b || ++b == 3); // c is true, b is 4
c = (a < b | ++b == 3); // c is true, b is 5
```

✿ Note: It is not advisable to mix boolean logic and actions in the same expression.



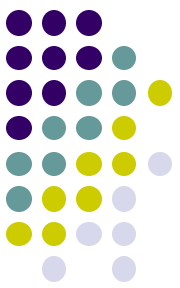
Other mathematical operations

Class `java.lang.Math` provides various mathematical operations:

```
double a = 1.99, b = 2.99, c = 0;
c = Math.cos(a);      // cosine
c = Math.acos(a);     // arc cosine
c = Math.sin(a);      // sine
c = Math.asin(a);     // arc sine
c = Math.tan(a);      // tangent
c = Math.atan(a);     // arc tangent
c = Math.exp(a);      // ea
c = Math.max(a,b);    // greater of two values
c = Math.min(a,b);    // smaller of two values
c = Math.pow(a,b);    // ab
c = Math.sqrt(a);     // square root
c = Math.random();    // random number 0.0<=c<1.0
```

Numeric rounding example :

```
int a = 11, b = 3;
long c = Math.round(a/b);           // c is 3
double d = Math.round(a/b);         // d is 3.0
double e = Math.round((double)a/b*100)/100.0; // e is 3.67
```

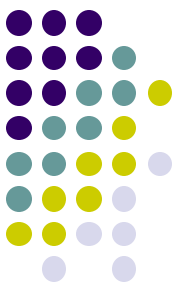



Control flow statements

if-else statement

```
if (<boolean expression>) {  
    // statements;  
} else {  
    // statements;  
}
```

```
if (<boolean expression>) {  
    // statements;  
} else if (<boolean expression>){  
    // statements;  
} else {  
    // statements;  
}
```



Control flow statements

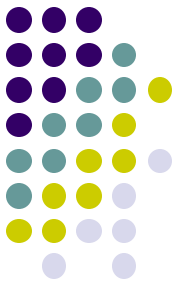
for statement

simple for

```
for (<initialization>; <boolean expression>; <increment>;) {  
    // statements;  
}
```

foreach

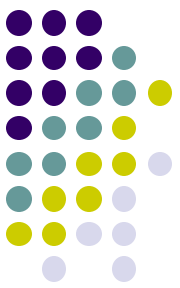
```
for (DataType varName: array | iterable collection){  
    // statements;  
}
```



Control flow statements

while statement

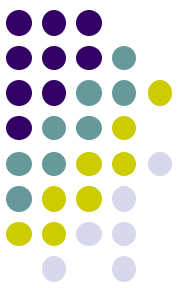
```
while(<boolean expression>) {  
    // statements;  
}
```



Control flow statements

do-while statement

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



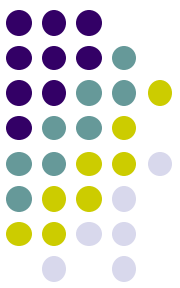
Control flow statements

switch statement

```
switch (expression) {  
    case value1:  
        // do something  
        break; // optional  
  
    case value2:  
        // do something else  
        break; // optional  
  
    ...  
    default: // optional  
        // do something if value is none of the cases above  
}
```

Limitations:

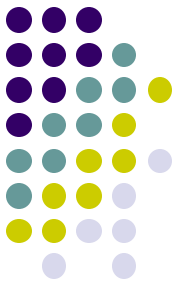
- expression can be one of types: boolean, integer type, string, enum.
- Duplicate case values are not allowed.
- Case values must be of the same type as the variable in the switch.
- The value for a case must be a constant or a literal
- break and default may be omitted



break and continue

```
int[ ] numbers = { 10, 20, 30, 40, 50 };  
  
int sum = 0;  
for (int x : numbers) {  
    if (x % 15 == 0) {  
        continue;  
    }  
  
    sum += x;  
  
    if (sum > 100) {  
        break;  
    }  
    System.out.println(x);  
}  
  
System.out.print("sum = " + sum);
```

Summary



- Understand the Java internals
- Describe primitives: bounds, Unicode, casting, overflow and underflow
- Operators: pre/postfix operators, short-circuiting, shorthand operators
- Control flow statements

Questions

