

Introduction and Overview

Introduction to Programming and
Computational Problem Solving:
Accelerated Pace

CSE 11

Lecture 1

CSE 11: Introduction to Programming and Computational Problem Solving: Accelerated Pace

- Today
 - Course overview
 - Logistics
 - Programs
 - Variables
 - Data types and literals
 - Introduction to objects and classes
 - Constructors

CSE 11 topics

Object-oriented programming

- Introduction to Java
- Numbers and mathematics
- Characters and strings
- Selections
- Methods
- Loops
- Recursion (simple)
- Arrays

Procedural programming

- Objects and classes
- Inheritance
- Polymorphism
- Abstract classes
- Interfaces
- Introduction to generics
- Exceptions
- File input/output
- Assertions

Procedural programming vs object-oriented programming

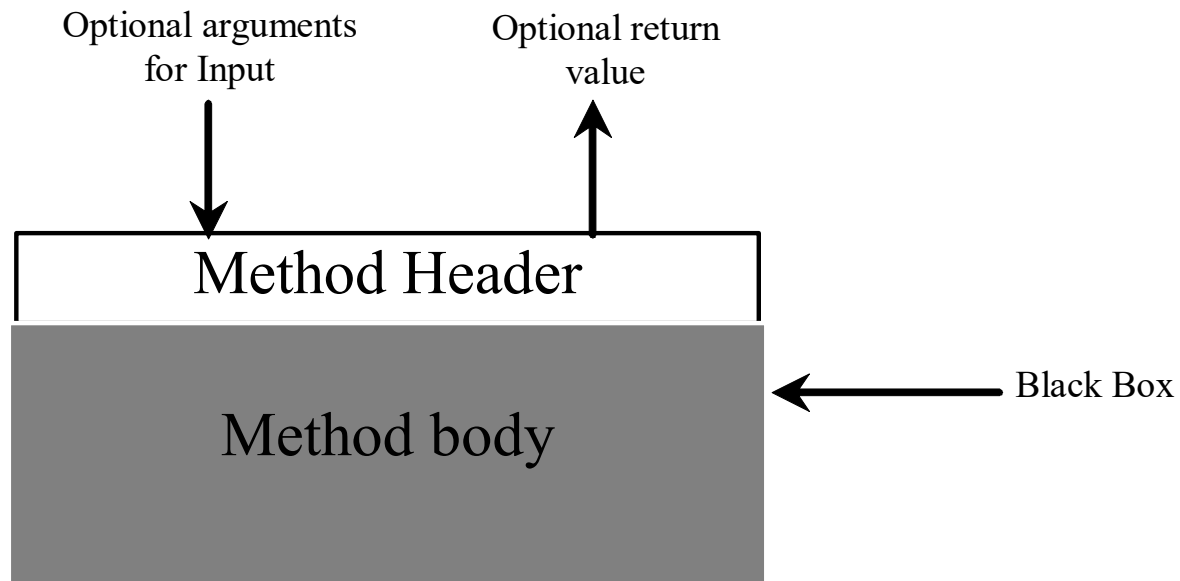
- Procedural programming
 - Procedural programming involves programming the steps a program must take to reach a desired state
 - This is the focus of CSE 8A
- Object-oriented programming
 - Object-oriented programming (OOP) involves programming using objects

Procedural programming vs object-oriented programming

- Procedural programming
 - Data and operations on data are separate
 - Requires passing data to methods
- Object-oriented programming
 - Data and operations on data are in an object
 - Organizes programs like the real world
 - All objects are associated with both attributes and activities
 - Using objects improves software reusability and makes programs easier to both develop and maintain

Procedural programming: method abstraction

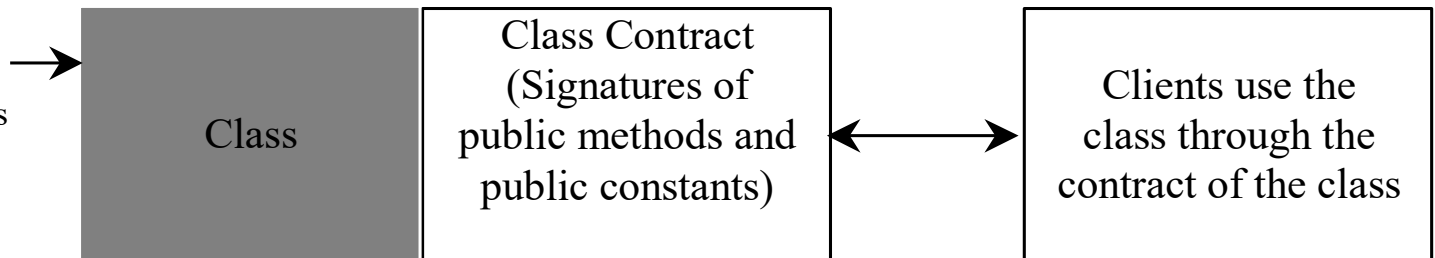
- You can think of the method body as a black box that contains the detailed implementation for the method



Object-oriented programming: class abstraction and encapsulation

- *Class abstraction* means to separate class implementation from the use of the class
- The creator of the class provides a description of the class and lets the user know how the class can be used
 - The *class contract*
- The user of the class does not need to know how the class is implemented
- The detail of implementation is encapsulated and hidden from the user
 - *Class encapsulation*
 - A class is called an *abstract data type* (ADT)

Class implementation
is like a black box
hidden from the clients



Syllabus

- Instructors: Ben Ochoa (Mon and Wed lecture, Thu discussion) and Joe Politz (Tue and Thu lecture, Wed discussion)
- TAs: Anya Bouzida, Daniela Perry, Gonzalo Allen-Perez, Neelkanth Shitolay, Rachel Lim, and Sydney Zhang
- Tutors: Amanda Tsai, Blake Newhouse, Brendan Barber, Gavin Wu, Kathy Charry, Marta Krylova, Tianlin Situ, and Zach Swanson
- Public course website <https://ucsd-cse11.github.io/fa26/>
 - Worksheets, notes, slides, etc.
 - Readings on PrairieLearn (book is on ucsd-cse11.github.io, but you must use PrairieLearn for credit)
 - Assignments on PrairieLearn (development of and/or submitting assignments)
 - Piazza for discussion
 - Gradescope to view completed worksheets
 - Course calendar for instructor office hours and TA/tutor help hours
 - See syllabus for course policies: AI use, academic integrity, in-class devices (sit in the back 2 rows if viewing non-lecture related content), collaboration, and student conduct
- Course is also on Canvas
 - Media gallery for recorded lecture and discussion meetings

Syllabus

- 19 (Ben) or 20 (Joe; Thu week 0 lecture) lecture meetings
 - No lecture on Veterans Day (Ben; Wed, Nov 11) and special topic lecture on Thu, Nov 12 (Joe)
 - Recorded lecture on Thanksgiving Eve (Ben; Wed, Nov 25; released Tue, Nov 24)
 - No lecture on Thanksgiving Day (Joe; Thu, Nov 26)
- Weekly discussion section meeting
 - Recorded discussion on Veterans Day (Joe; Wed, Nov 11)
 - Recorded discussion on Thanksgiving Day (Ben; Thu, Nov 26)

Syllabus

- Grading (see syllabus on course website for more details and grading policy)
 - Social learning and engagement
 - Readings, and lecture and discussion meetings
 - 6 assignments
 - Initial deadline tentatively Fridays in weeks 1, 2, 4, 6, 8, and 10 (grade of 0, 1, 2, 3, or 4)
 - Not accepted late, must upgrade 0 via Review/Resubmission
 - Review/Resubmission deadline about 2 weeks later (0 or 1 upgraded to 3 maximum; 2 or 3 upgraded to 4 maximum)
 - 3 exams
 - 45 minute exams during weeks 3, 6, and 9 at the Triton Testing Center's (TTC's) Computer-Based Testing Facility (CBTF)
 - Can retry any or all exams in a 2-hour period during finals week at the TTC's CBTF
 - PrairieTest to make reservations for weeks 3, 6, 9, and finals
 - Students with approved accommodations via an Authorization for Accommodation (AFA) letter issued by the Office for Students with Disabilities (OSD) will be taking their exams at the TTC, where exams must be scheduled at least 72 hours in advance

Programs and Values

Introduction to Programming and
Computational Problem Solving:
Accelerated Pace

CSE 11

Lecture 1

Programs

- Computer programs (i.e., software) are instructions to the computer
- You tell a computer what to do through programs
- Computers do not understand human languages, so you need to use computer languages to communicate with them
- Programs are written using programming languages

Programming languages

- High-level language
 - High-level languages are English-like and easier to learn and program than assembly languages
 - For example, the following is a high-level language statement that computes the area of a circle with radius 5:
`area = 5 * 5 * 3.1415;`

Introduction to Java

- Java is:
 - a high-level programming language
 - Computer-specific details are abstracted
 - an object-oriented programming language
 - Based on classes
 - a strongly typed language
 - **Programmers must explicitly identify the type of every variable, method, and object**

Naming conventions

- Variables: Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name
 - For example, the variables `radius` and `isValid`
- Classes: Capitalize the first letter of each word in the name class (e.g., `WonderWall`)
 - Class names are nouns

Variables

- Variables are used to represent values that may be changed in the program

```
// Compute the first area
```

```
radius = 1.0;
```

```
area = radius * radius * 3.14159;
```

```
// Compute the second area
```

```
radius = 2.0;
```

```
area = radius * radius * 3.14159;
```

Declaring variables

```
int x;           // Declare x to be an
                  // integer variable

double radius;  // Declare radius to
                  // be a double variable

char a;         // Declare a to be a
                  // character variable
```

Assignment statements

```
x = 1;           // Assign 1 to x  
radius = 1.0;    // Assign 1.0 to radius  
a = 'A';         // Assign 'A' to a
```

Declaring and initializing in one step

```
int x = 1;
```

```
double radius = 1.0;
```

```
char a = 'A';
```

Numerical data types

Name	Range	Storage Size
<code>byte</code>	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
<code>short</code>	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
<code>int</code>	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
<code>long</code>	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
<code>float</code>	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
<code>double</code>	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Number literals

- A literal is a constant value that appears directly in the program

```
int i = 34;  
long x = 1000000;  
double d = 5.0 + 1.0;
```

34, 1000000, 5.0,
and 1.0 are literals

Integer literals

- An integer literal can be assigned to an integer variable as long as it can fit into the variable
- A compilation error will occur if the literal is too large for the variable to hold
 - For example, the statement `byte b = 1000` would cause a compilation error, because `1000` cannot be stored in a variable of the `byte` type
- An integer literal is assumed to be of the `int` type, whose value is between -2^{31} (equals -2147483648) to $2^{31}-1$ (equals 2147483647)
- To denote an integer literal of the `long` type, append it with the letter `L` or `l`
 - `L` is preferred because `l` (lowercase `L`) can easily be confused with `1` (the digit one)

Floating-point literals

- Floating-point literals are written with a decimal point
- By default, a floating-point literal is treated as a `double` type value
 - Example: `5.0` is considered a `double` value, not a `float` value
- You can make a number a `float` by appending the letter `f` or `F`, and make a number a `double` by appending the letter `d` or `D`
 - Example: you can use `100.2f` or `100.2F` for a float number, and `100.2d` or `100.2D` for a double number

double vs float

- The double type values are more accurate than the float type values

– For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays 1.0 / 3.0 is 0.3333333333333333

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays 1.0F / 3.0F is 0.33333334

7 digits

Floating-point accuracy

- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy
- For example,
`System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);`
displays `0.50000000000000000001`, not `0.5`, and
`System.out.println(1.0 - 0.9);`
displays `0.09999999999999999998`, not `0.1`
- Integers are stored precisely
 - Calculations with integers yield a precise integer result

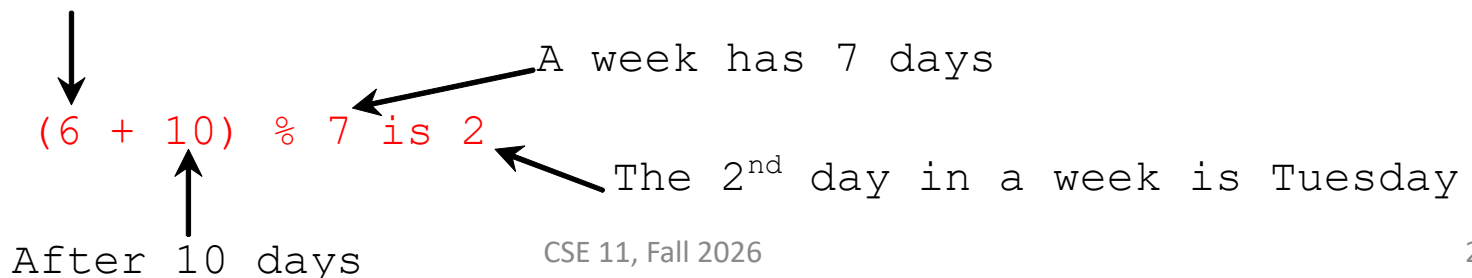
Integer division

- Warning: resulting fractional part (i.e., values after the decimal point) are **truncated, not rounded**
 - Example: $5 / 2$ yields an integer 2

Remainder operator

- `dividend % divisor`
- The sign of the remainder is the sign of the dividend
- Example: an even number `% 2` is always 0 and an odd number `% 2` is always 1
 - You can use this property to determine whether a number is even or odd
- Example: If today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression.

Saturday is the 6th day in a week



Scientific notation

- Floating-point literals can also be specified in scientific notation
 - Example: $1.23456e+2$ (same as $1.23456e2$) is equivalent to 123.456 , and $1.23456e-2$ is equivalent to 0.0123456
- E or e represents an exponent

Conversion rules

- When performing a binary operation involving two operands of *different* types, Java automatically converts the operand based on the following rules
 1. If one of the operands is `double`, the other is converted into `double`
 2. Otherwise, if one of the operands is `float`, the other is converted into `float`
 3. Otherwise, if one of the operands is `long`, the other is converted into `long`
 4. Otherwise, both operands are converted into `int`

Type casting

Implicit casting

```
double d = 3; (type widening)
```

Explicit casting

```
int i = (int)3.0; (type narrowing)
```

```
int i = (int)3.9; (fraction part is truncated, not rounded!)
```

range increases



byte, short, int, long, float, double

char data type

Character literal

```
char letter = 'A'; // ASCII
char numChar = '4'; // ASCII
char letter = '\u0041'; // Unicode
char numChar = '\u0034'; // Unicode
```

- Java characters use Unicode, a 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world's diverse languages
- Unicode takes two bytes, preceded by `\u`, expressed in four hexadecimal numbers that run from `\u0000` to `\uFFFF`
 - Unicode can represent 65536 characters

Common and special characters

Characters	Code Value in Decimal	Unicode Value
'0' to '9'	48 to 57	\u0030 to \u0039
'A' to 'Z'	65 to 90	\u0041 to \u005A
'a' to 'z'	97 to 122	\u0061 to \u007A

<i>Escape Sequence</i>	<i>Name</i>	<i>Unicode Code</i>	<i>Decimal Value</i>
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed	\u000A	10
\f	Formfeed	\u000C	12
\r	Carriage Return	\u000D	13
\\	Backslash	\u005C	92
\"	Double Quote	\u0022	34

Casting between char and numeric data types

```
int i = 'a'; // Same as int i = (int)'a';
```

```
char c = 97; // Same as char c = (char)97;
```

String type

- The char type only represents one character
- To represent a string of characters, use the String type
- String is a predefined class in the Java library (just like the System class and Scanner class)

<https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/lang/String.html>

String type

- `String` is a predefined class in the Java library

```
String message = "Welcome to Java";
```

String literal



- The `String` type is not a primitive type; it is known as a reference type
 - Any Java class can be used as a reference type for a variable

Conversion between strings and numbers

- String to number

```
int intValue = Integer.parseInt(intString);  
double doubleValue =  
    Double.parseDouble(doubleString);
```

- Number to string

```
String s = number + "";  
String intString =  
    Integer.toString(intValue);  
String doubleString =  
    Double.toString(doubleValue);
```

The boolean type and operators

- Often in a program you need to compare two values, such as whether `i` is greater than `j`
- Java provides six comparison operators (also known as relational operators) that can be used to compare two values
- The result of the comparison is a Boolean value: `true` or `false`
- For example

```
boolean b = (1 > 2);
```

Relational operators

Java Operator	Mathematics Symbol	Name	Example (radius is 5)	Result
<	<	less than	<code>radius < 0</code>	<code>false</code>
<=	≤	less than or equal to	<code>radius <= 0</code>	<code>false</code>
>	>	greater than	<code>radius > 0</code>	<code>true</code>
>=	≥	greater than or equal to	<code>radius >= 0</code>	<code>true</code>
==	=	equal to	<code>radius == 0</code>	<code>false</code>
!=	≠	not equal to	<code>radius != 0</code>	<code>true</code>

Logical operators

Operator	Name	Description
!	not	logical negation
&&	and	logical conjunction
	or	logical disjunction
^	exclusive or (xor)	logical exclusion

Short-circuit operators

- `&&` and `||` are short-circuit operators
- `p1 && p2`
 - If `p1` or `p2` is false, then `p1 && p2` is false
 - `p1` is evaluated first
 - If `p1` is true, then `p2` is evaluated
 - If `p1` is false, then `p2` is **not** evaluated
- `p1 || p2`
 - If `p1` or `p2` is true, then `p1 || p2` is true
 - `p1` is evaluated first
 - If `p1` is true, then `p2` is **not** evaluated
 - If `p1` is false, then `p2` is evaluated

Operator precedence

- `()`, `var++`, `var--`
- `++var`, `--var`, `+`, `-` (unary plus and minus), `!` (not)
- (type) casting
- `*`, `/`, `%` (multiplication, division, and remainder)
- `+`, `-` (binary addition and subtraction)
- `<`, `<=`, `>`, `>=` (relational operators)
- `==`, `!=` (equality)
- `^` (exclusive or)
- `&&` (and)
- `||` (or)
- `=`, `+=`, `-=`, `*=`, `/=`, `%=` (assignment operators)

Operator associativity

- When two operators with the same precedence are evaluated, the associativity of the operators determines the order of evaluation
- All binary operators except assignment operators are left-associative
 - $a - b + c - d$ is equivalent to $((a - b) + c) - d$
- Assignment operators are right-associative
 - $a = b += c = 5$ is equivalent to $a = (b += (c = 5))$

Operator precedence and associativity

- The expression in the parentheses is evaluated first
 - Parentheses can be nested, in which case the expression in the inner parentheses is executed first
- When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule
- If operators with the same precedence are next to each other, their associativity determines the order of evaluation

Objects and classes

- An object represents an entity in the real world that can be distinctly identified
 - For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects
 - An object has a unique identity, state, and behaviors
- Classes are constructs that define objects of the same type

Objects

- An object has a unique identity, state, and behaviors
 - An object is a **unique instance of a class**
 - The **state** of an object consists of a **set of data fields** (also known as properties) with their current values
 - The **behavior** of an object is defined by a **set of methods**

Classes

- A Java class uses **variables to define data fields** and **methods to define behaviors**
- Additionally, a class provides a special type of methods, known as **constructors**, which are invoked to construct objects from the class

Constructors

- Constructors must have the **same name** as the class itself
- Constructors are invoked using the new operator when an object is created
- Constructors play the role of initializing objects

Declaring object reference variables

- To reference an object, assign the object to a **reference variable**
- To declare a reference variable, use the syntax
`ClassName objectRefVar;`
- For example
`Circle circle;`

Declaring and creating in one step

```
ClassName objectRefVar =  
    new ClassName(argument1, argument2, ...);
```

For example

Assign object reference Create an object

```
Circle myCircle = new Circle(20, "red");
```

Next lecture

- Records and methods