Introduction to Java

Faraaz Sareshwala

Java

- Object Oriented
- Write once, run anywhere
- Compiled, reasonable performance
- Why Java?
  - One language for the entire class
  - Textbook is targeted towards Java

Hello World!

```
package main;


public class Main {

  public static void main(String[] args) {

    System.out.println("Hello World!");

  }

}
```

- All methods, variables, etc must be contained within a class
- Each class must be in a file with the same name
  - File extension: .java
  - Class Main needs to be in Main.java

Primitive Types

Description	Type	Size
Integer	byte	1 byte
	short	2 bytes
	int	4 bytes
	long	8 bytes
Decimal	float	4 bytes
	double	8 bytes
Other	boolean	1 bit
	char	Unicode

Declaring Variables

- Must declare a variable before you can use it
- Variables must declare their type
- Variables cannot change type after being declared

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

  int x = 0;

  x = "Foo"; // wrong

}
```

Standard Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
( )	Parenthesis
!	Negation
==	Equality
!=	Inequality

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

  int x = 0;

  int y = (x + 2) * 2;

  boolean even = (y % 2);

  boolean value = (y != 0);

}
```

Converting Types

- C-style typecasting (unsafe)

```
long x = Long.MAX_VALUE;

int y = (int) x; // data loss
```

- [Type]Value methods

```
Integer x = 5;

double y = x.doubleValue();

double y = (double) x.intValue();
```

- Static helper methods
  - valueOf
  - parse[Type]

```
String value = String.valueOf(y); // int to String

int z = Integer.parseInt(value); // String to int
```

If Statements

```
if (boolean expression) {
  statements;
}
else if (boolean expression) {
  statements;
}
else {
  statements;
}
```

```
if (middle < 5) {
  high = middle - 1;
}
else if (middle > 5) {
  low = middle + 1;
}
else {
  middle = 0;
}
```

While Loop

```
while (boolean expression) {

  statements;

}
```

```
int i = 0;


while (i < 100) {

  if (i % 2 == 0) {

    System.out.println(i);

  }


  i++;

}
```

For Loop

```
for (init; boolean expression; update) {
  statements;
}
```

```
for (int i = 0; i < 100; i++) {
  if (i % 2 == 0) {
    System.out.println(i);
  }
}
```

Loop Iteration Control

Break ends the enclosing loop

```
for (int i = 0; i < 100; i++) {
  if (i % 2 == 0) {
    System.out.println(i);
    break;
  }
}
```

Continue immediately starts the next iteration

```
for (int i = 0; i < 100; i++) {
  if (i % 2 == 0) {
    continue;
    System.out.println(i);
  }
}
```

Arrays

- Contiguous sequences of memory
- All elements must be of the same type
- Arrays know their own size
- Content zero'd out by default

```
int[] array = new int[10]; // 10 zeros

String[] strings = new String[5]; // 5 nulls


int[] elements = {1, 2, 3, 4};


System.out.println(array[2]);

System.out.println(elements.length);
```

Looping Through Arrays

Standard for loop

```
int[] array = new int[10];

for (int i = 0; i < array.length; i++) {

  int element = array[i];

  // access array[i]

}
```

Foreach loop

```
int[] array = new int[10];

for (int element : array) {

  // access element

}
```

Classes

- All code must be within a class
- Each instance variable or method is prefixed with a visibility modifier
  - Public
  - Private
  - Protected
- Each class must be in a file of the same name

```
public class Rectangle {

  private int length = 0;

  private int width = 0;

}
```

Instance Methods

- Functions that operate on the class
- Must declare return type
- Should include access modifier

```
public class Rectangle {

  public int getArea() {

    return length * width;

  }

}
```

More About Methods

- Parameter lists must include type of each argument
- No default parameters
  - Use method overloading instead
- Void methods return nothing
- Non-void methods must return a value via every path through the method

```
public int foo(int x) {
  return foo(x, 10);
}
```

```
public int foo(int x, int y) {
  if (x < 10) {
    return 0;
  }
  return x + y;
}
```

Constructors

- Constructors run on each instantiation (object creation) of a class
- No return type
- Must have same name as the class
- Usually has public access modifier
- Can have multiple constructors

```
public class Rectangle {

  private int length = 0;

  private int width = 0;


  public Rectangle() { }

  public Rectangle(int length, int width) {

    this.length = length;

    this.width = width;

  }

}
```

Getters and Setters

- Good practice to keep all instance variables private
- Grant access through getters and setters
- Can keep validation logic in a single place

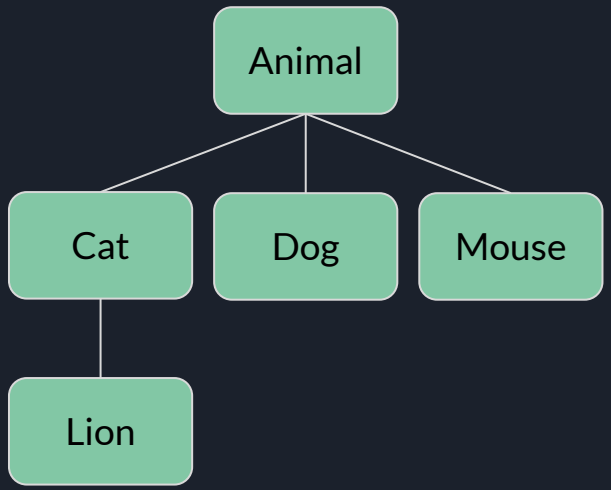
```
public class Rectangle {

  public int getLength() {

    return length;

  }


  public void setLength(int length) {

    if (length > 0) {

      this.length = length;

    }

  }

}
```

Example Usage of Rectangle Class

```
public class Main {

  public static void main() {

    Rectangle rectangle = new Rectangle(10, 5);

    System.out.println(rectangle.getLength());

    rectangle.setWidth(10);

    System.out.println(rectangle.getArea());

  }

}
```

Inheritance and Polymorphism

- Java supports only single inheritance using the `extends` keyword




```java
public class Animal { }
```

```java
public class Cat extends Animal { }

public class Dog extends Animal { }

public class Mouse extends Animal { }
```

```java
public class Lion extends Cat { }
```

Multiple Inheritance of Interfaces

Java “supports” multiple inheritance through interfaces

```
public interface Player {
  void move();
}

public interface Admin {
  void ban(Player player);
}

public class SuperUser extends User implements Player, Admin {
  void move() { ... } // required from interface Player
  void ban(Player player) { ... } // required from interface Admin
}
```

Generics

- Generics allow for templated, reusable code
- Can have multiple types

```
public class List<T> {
  T value;
  T getValue() { return value; }
}

public class Map<Key, Value> { ... }

public class Main {
  public static void main(String[] args) {
    List<Integer> list = new List<>();
    Map<String, Boolean> map = new Map<>();
  }
}
```