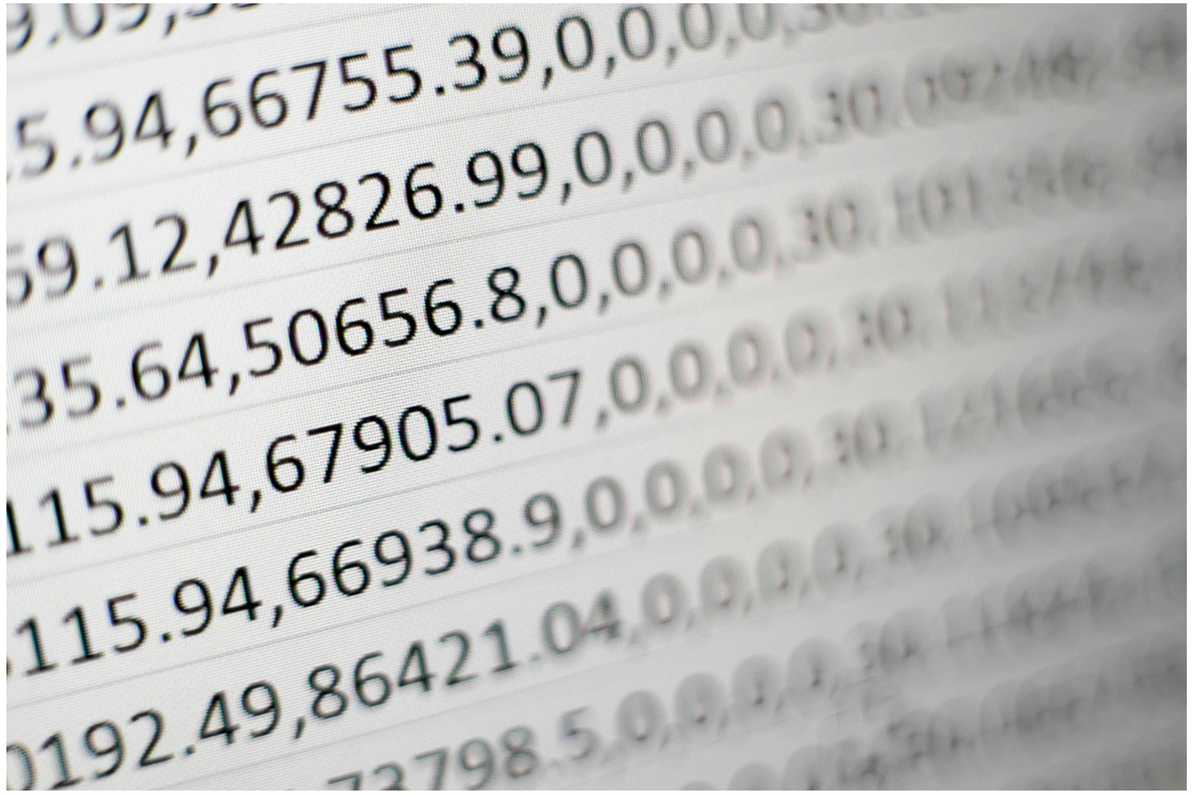
Plain Old Java Objects (POJO)

DATA TYPES AND EXCEPTIONS IN JAVA

Jim White
Java Developer

Data types and exception handling overview

- Chapter 1
  - Plain old (or ordinary) Java objects
  - Wrapper classes
  - Using packages
- Chapter 2
  - Collections Framework
- Chapter 3
  - Exceptions and Exception handling



[1] Photo by Mika Baumeister on Unsplash

Java objects as data structures

- Java objects store and organize data.
  - Created from custom classes
- Store data in fields
  - Act as application data "suitcases"
- Called **POJOs**
  - **Plain ordinary Java object**
  - Or Plain old Java object
- Simple Java objects hold and move data
  - Conform to some rules
  - Don't have any logic

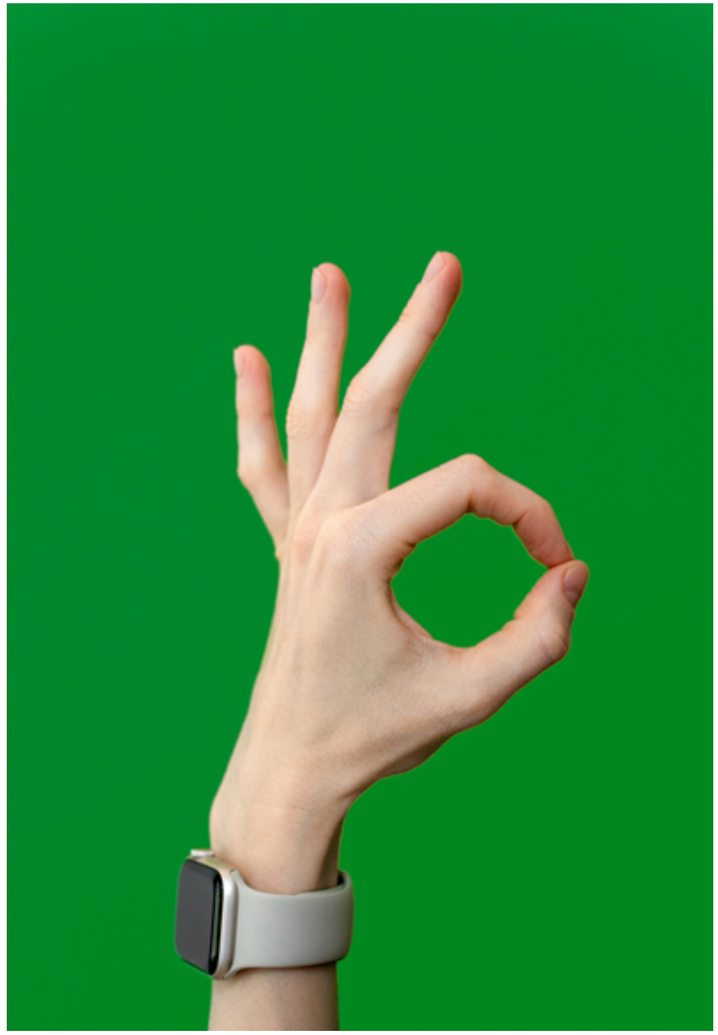
[1] Photo by Caroline Selfors on Unsplash

Getters and setters

- Getters and Setters are public
  - Fields are private
- Getter and Setters protect the data in a POJO
  - Promote data encapsulation
- Hide the implementation details of underlying fields

POJO class guidelines

- POJO class **should:**
  - Be public
  - Have public getters/setters for all fields
    - Fields should be private
  - Have a default (no argument) constructor



[1] Photo by Anastasiya Badun on Unsplash

POJO don'ts

- POJO class **should not:**
  - Be tied to a framework
  - Extend other classes
  - Implement any interface
    - Other than Serializable in some cases
  - Contain business logic
- Even these rules are sometimes relaxed
  - Try to keep POJOs simple

[1] Photo by Anastasiya Badun on Unsplash

Getters

- Getter method names
  - Start with "get" (boolean getters start with "is")
  - End with the field name
  - Lower camel case
  - Example: `get` + `make` = `getMake`
- Take no parameters
- Return the field value
- Getters can hide details about field implementation

```
private String make;

public String getMake() {
    return make;
}
```

```
private short on;

public boolean isOn() {
  if (on = 0) {
    return false;
  } else {
    return true;
  }
}
```

Setters

- Setter method names
  - Start with "set", end with the field name
  - Lower camel case
  - Example: `set` + `make` = `setMake`
- Take a single parameter - new value of the field
  - Use `this` to distinguish field from parameter
- Return void (nothing)
- Can check the validity of the data

```
private String make;

public void setMake(String make) {
    this.make = make;
}
```

```
private int age;

public void setAge(int age) {
  if ((age >= 0) && (age <= 120)) {
    this.age = age;
  }
}
```

POJO example

- Public class
- Private fields
- Public getters/setters
  - Getters return the value of a field
  - Setters set the value of a field
- Default / no argument constructor

```
public class Car {   // POJO class is public
    private String model;  // Fields are private
    private int year;

    // Default no arg constructor

    // Public getters to access POJO data
    public String getModel() {
        return model;
    }
    public int getYear() {
        return year;
    }
    // Public setters to set POJO fields
    public void setModel(String model) {
        this.model = model;
    }
    public void setYear(int year) {
        this.year = year;
    }
}
```

Let's practice!

DATA TYPES AND EXCEPTIONS IN JAVA

Wrapper classes

DATA TYPES AND EXCEPTIONS IN JAVA

Jim White
Java Developer

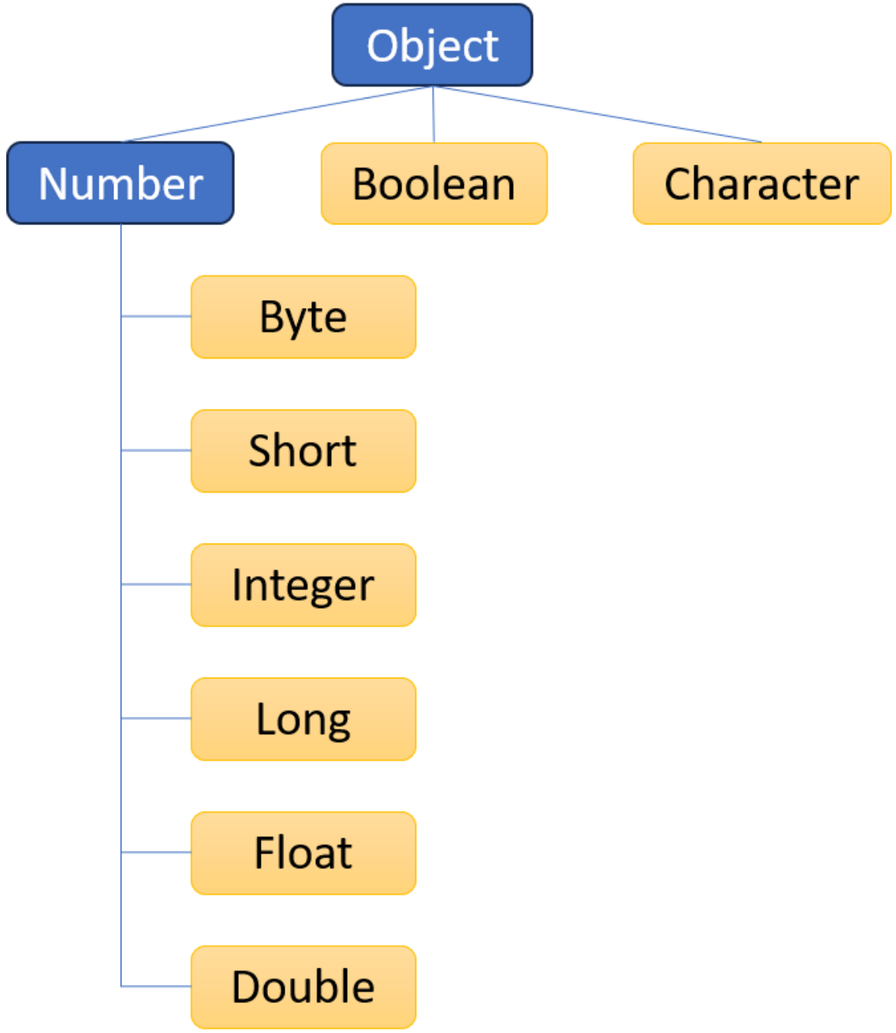
Java 8 primitives

	Primitive type	Value	Example
whole numbers	**byte**	-128 to 128	`byte age = 5;`
	short	-32,768 to 32,767	`short sqFeet = 3240;`
	int	-2,147,483 and 2,147,647	`int accountBalanace = 100000;`
	long	-2^63 and 2^63	`long nationalDebt = 34849081491943L;`
decimals	**float**	3.4e-038 to 3.4e+038	`float unitPrice = 4.99f`
	double	1.7e-308 to 1.7e+308	`double pi = 3.141592653589793;`
	boolean	**true** or **false**	`boolean is Zoned = true;`
	char	alphabet, punc, numbers, etc.	`char grade = 'C';`

Wrapper classes

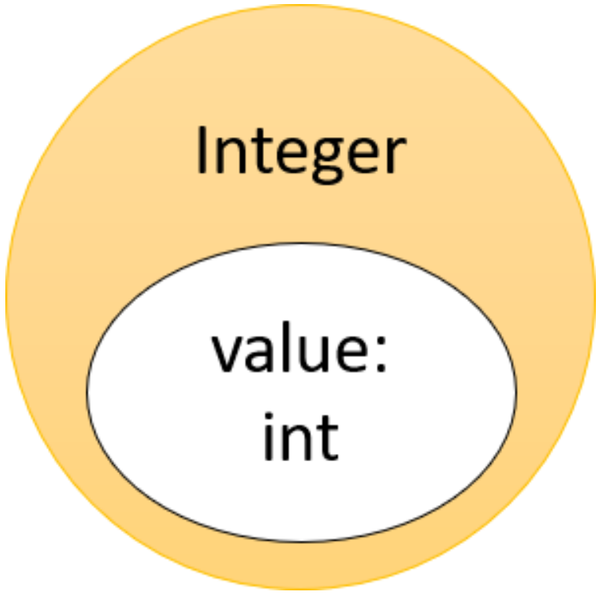
- Each primitive type has an equivalent wrapper class

primitive	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character



Wrapping primitives

- Instances of a wrapper class hold a single primitive
  - Example: `Integer` holds a single `int` primitive value



Wrapper details

- Wrapper classes have additional fields and methods
  - Fields (like max value) about the type
  - Methods to work with the wrapped primitive type
- Wrapper classes are in base language
  - Use them without `import` (more on imports later)

Creating wrapper objects

- Assign a primitive to a variable of the wrapper type
  - Syntax: `Wrapper-type variable = primitive-value;`

```
Integer age = 12;
Double cost = 150250.55;
Float interest = 5.5f;
Character grade = 'A';
Boolean isActive = true;
```

- Wrapper objects can also have no value (`null`)

```
Integer age = null;
```

Using wrapper objects

```
Integer age = 12;
```

- Print them

```
System.out.println(age);    // Displays 12
```

- Get their primitive value

```
int x = age.intValue();     // x is assigned 12
```

- Perform other operations on them

```
double z = age.doubleValue();  //z is 12.0
String y = age.toString();  // y is assigned "12"
Integer teenAge = 16;
int smaller = age.compareTo(teenAge);  // smaller is -1 since 12 < 16
```

Wrapper Static Methods

- Wrapper classes come with static methods
- Used to perform operations on the associated primitives

```
int x = Integer.sum(8,12);  // x is 20
int y = Integer.remainderUnsigned(102, 10);  // y is 2
```

- Used to convert between String and primitive

```
int z = Integer.parseInt("123");   // z is 123;
boolean ans = Boolean.parseBoolean("false");  // ans is false
```

[1] See https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html for more on the Integer wrapper

Wrapper Static Fields

Example static fields on wrapper classes

```
System.out.println(Integer.MAX_VALUE); // Maximum value an int can have
System.out.println(Integer.MIN_VALUE); // Minimum value an int can have
```

```
2147483647
-2147483648
```

```
System.out.println(Boolean.TRUE); // Corresponding to the primitive value of true
System.out.println(Boolean.FALSE); // Corresponding to the primitive value of false
```

```
true
false
```

Wrapper Static Fields

Example static fields on wrapper classes

```
System.out.println(Character.SPACE_SEPARATOR);  // Unicode for the regular space like ' '
System.out.println(Character.LINE_SEPARATOR); // Unicode for line return like '\n'
```

```
12
13
```

Interesting Wrapper Methods

Wrapper Method	Returns
Boolean.logicalAnd(boolean a, boolean b)	boolean
Boolean.logicalOr(boolean a, boolean b)	boolean
Boolean.parseBoolean(String s)	boolean
Character.getNumericValue(char ch)	Unicode int value of the char
Character.isDigit(char ch)	boolean
Character.isLowerCase(char ch)	boolean
Character.isWhitespace(char ch)	boolean
Double.parseDouble(String s)	double
Double.longValue()	the Double value, rounded down, as a Long

Why Wrappers?

- Primitives have no methods (only operations)
  - Wrapper classes come with useful methods

```
int score = Integer.parseInt("8");
```

- Wrappers allow primitives to be used as objects; like putting them in a collection
  - Learn about collections in the Chapter 2.

Wrappers can be null

- Wrappers allow instance or static variables of the type to be `null`
  - Primitive instance or static variables have a default value when not initialized

```
int herAge; // age is 0 by default
Integer hisAge = null;
if (hisAge != null) {
  // do something when hisAge is not set
}
```

Let's practice!

DATA TYPES AND EXCEPTIONS IN JAVA

Using packages

DATA TYPES AND EXCEPTIONS IN JAVA

Jim White
Java Developer

What are Java packages?

- Packages organize Java code
  - A package acts like a file folder or directory
  - Combining related Java code (classes, interfaces, enums, etc.)
- Like file folders
  - Each package has a name

[1] Image from https://commons.wikimedia.org/wiki/File:File_Cabinet.jpg

Types of packages

- **Built-in packages**
  - Part of Java
  - Name begins with "java" or "javax"
- **User defined packages**
  - Packages we define
  - Packages we get from 3rd parties

Package names

- Package names follow a convention
  - Written in all lowercase
  - Use periods (`.`) to delimit name parts
- Built-in packages
  - Start with `java` or `javax`
  - Rest of the name suggests purpose
- User defined package names
  - Start with the reverse of the organization's domain
  - Rest of the name suggests functionality

- Example built-in packages

```
java.security
java.time
```

- Example user-defined package names

```
com.mycompany.myproject
com.mycompany.myproject.account
com.mycompany.myproject.controller
com.mycompany.myproject.ui
```

Built-in packages

Some commonly used built-in packages

Package	Contains/Provides
`java.lang`	Base language support classes
`java.io`	Input / output operations
`java.util.logging`	Logging framework
`java.math`	Precision integer and decimal arithmetic
`java.net`	Networking operations
`java.util`	Date / time and data structures like Linked List, Dictionary and support
`java.security`	Security framework

java.math

- `java.math` provides classes for arithmetic
  - Used in cryptographic, scientific, and currency/money applications
- `BigInteger` for representing large integers
  - Integers larger than `int` or `long` can handle
  - To work with integers of almost unlimited number of digits.
- `BigDecimal` for representing very large or small floating point number
  - Deals with rounding errors that can occur with `float` or `double`

Using packages

- Use `import` + package name at the top

```
import java.math.BigInteger;
public class HelloWorld {
  BigInteger acct = new BigInteger("123");
}
```

- Packages can contain many types.
  - Use `*` to import all types in the package

```
import java.math.*;
public class HelloWorld {
  BigInteger acct = new BigInteger("123");
  BigDecimal pi = new BigDecimal("3.14");
}
```

BigInteger & BigDecimal from java.math

- `BigInteger` and `BigDecimal` act as wrappers for big numbers
- Construct them using a `String` or numeric
- Come with `add`, `subtract`, `multiply`, and `divide` methods
- Have additional methods like `pow` for power

```
// Imports go at the top of the class
import java.math.BigInteger;
import java.math.BigDecimal;

// Create BigInteger or BigDecimal with String
BigInteger big = new BigInteger("1000");
BigInteger ten = new BigInteger("10");
BigDecimal pi = new BigDecimal("3.14");
// Using a primitive to create BigDecimal
BigDecimal one = new BigDecimal(1.0);
```

```
BigInteger x = big.add(ten); // = 1010
BigDecimal y = pi.add(one); // = 4.14
BigInteger bigSqr = big.pow(2); // = 1000000
BigDecimal piCubed = pi.pow(3); // = 30.959144
```

BigInteger and BigDecimal methods

Method	Description
abs()	Absolute value of the integer
add(x)	Add x to the integer or decimal
divide(x)	Divide the integer or decimal by x
multiply(x)	Multiply the integer or decimal by x
negate()	Negate the integer or decimal
pow(int x)	The integer or decimal to the power of x
subtract(x)	Subtract x from the integer or decimal

[1] See https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html and https://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html

Import not required

- **Importantly** - `java.lang` package is imported automatically.
  - When using anything from `java.lang` it does not require an import
- `java.lang` includes `System`, `String`, the wrapper classes and `Exception`.
  - Explains why we can learn and use the base language elements without seeing an import.

Let's practice!

DATA TYPES AND EXCEPTIONS IN JAVA