Programming in Java – Day 3 Recap

Simple and complex data types, more on branches

School of Computing and Mathematical Sciences
Birkbeck, University of London

Topics on Day 3

- Static vs dynamic typing
- Simple data types
- Complex data types: classes, enums
- Strings, boxed types
- Branching: **`switch`**, the ternary operator

Static vs dynamic typing

- Type of a variable: says what kind of data the “box” can store

Static vs dynamic typing

- Type of a variable: says what kind of data the “box” can store
- Dynamic typing: the type of a variable may change over time

Static vs dynamic typing

- Type of a variable: says what kind of data the “box” can store
- Dynamic typing: the type of a variable may change over time
- Static typing: the type of a variable is fixed

Static vs dynamic typing

- Type of a variable: says what kind of data the “box” can store
- Dynamic typing: the type of a variable may change over time
- Static typing: the type of a variable is fixed
- Java uses **static** typing – Java needs to know what you will want to put into your boxes

```
int i;      // box only for int data
String s;   // box only for String data
```

Simple types

- Also called “primitive types”

Simple types

- Also called “primitive types”
- Java has eight of them:
  `int`, **`boolean`**, **`char`**, **`long`**, **`double`**, **`float`**, **`short`**, **`byte`**

Simple types

- Also called “primitive types”
- Java has eight of them:
  `int`, **`boolean`**, **`char`**, **`long`**, **`double`**, **`float`**, **`short`**, **`byte`**
- Represented efficiently in hardware

Simple types for integer numbers

- `long` uses 64 bits: values from -2^{63} to $2^{63} - 1$
- `int` uses 32 bits: values from -2^{31} to $2^{31} - 1$
- `short` uses 16 bits: values from -2^{15} to $2^{15} - 1$
- `byte` uses 8 bits: values from -2^{7} to $2^{7} - 1$

Simple types for integer numbers

- **long** uses 64 bits: values from -2^{63} to $2^{63} - 1$
- **int** uses 32 bits: values from -2^{31} to $2^{31} - 1$
- **short** uses 16 bits: values from -2^{15} to $2^{15} - 1$
- **byte** uses 8 bits: values from -2^{7} to $2^{7} - 1$
- Predefined operations: `+`,`-`,`*`,`/`,`%`,
- Predefined comparisons: `==`,`!=`,`>`,`>=`,`<`,`<=`,...

Simple types for floating-point numbers: **double** and **float**

- Good for (decimal) fractions: can hold values like 1.5
- **double** uses 64 bits, **float** only 32 bits
- Beware: rounding errors (wrt rational/real numbers from Maths)

```
double d1 = (0.1+0.1)/0.3;
double d2 = 2.0* 1000.0/9000.0*3.0;
// WRONG!
if (d1 == d2) {
    // this is not printed due to rounding errors
    System.out.println("Exactly the same (wrong comparison)");
}
// RIGHT!
if (Math.abs(d1 - d2) < 10E-6) {
    System.out.println("About the same (right comparison)");
}
```

Simple type for truth values: **`boolean`**

- Values **`true`** and **`false`**
- In Java: **`boolean`** is **not** an integer type

Simple type for characters: **char**

- Uses 16 bits

Simple type for characters: **char**

- Uses 16 bits
- Literals in Java using single quotes: `'a'`

Simple type for characters: **char**

- Uses 16 bits
- Literals in Java using single quotes: `'a'`
- Also an integer-like type: `'a' + 1 == 'b'`

Simple type for characters: **char**

- Uses 16 bits
- Literals in Java using single quotes: `'a'`
- Also an integer-like type: `'a' + 1 == 'b'`
- Ingredients for Strings

Complex types

How about “large” data, like Strings?

Complex types

How about “large” data, like Strings?

Complex types are stored in two parts:

- The data is in a special shared memory area: the **heap**
- Box on **stack** stores **pointer** (or **reference**) to the heap with memory address

Complex types

How about “large” data, like Strings?

Complex types are stored in two parts:

- The data is in a special shared memory area: the **heap**
- Box on **stack** stores **pointer** (or **reference**) to the heap with memory address

```
String str = new("This is a String!");
```

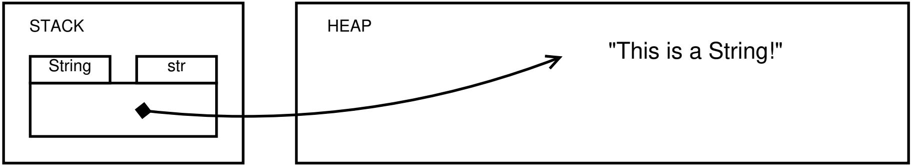
Complex types

How about “large” data, like Strings?

Complex types are stored in two parts:

- The data is in a special shared memory area: the **heap**
- Box on **stack** stores **pointer** (or **reference**) to the heap with memory address

```
String str = new("This is a String!");
```



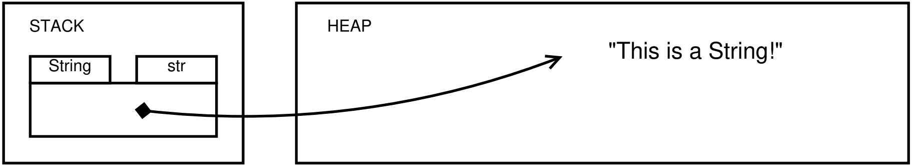
Complex types

How about “large” data, like Strings?

Complex types are stored in two parts:

- The data is in a special shared memory area: the **heap**
- Box on **stack** stores **pointer** (or **reference**) to the heap with memory address

```
String str = new("This is a String!");
```



Heap: memory reserved at runtime, while the program is executed
→ less efficient, more flexible than boxes for simple types

null pointers

What if we haven't reserved memory yet?

```
String str = null; // pointer to no data
int n = str.length(); // what happens?
```

A popular complex type: String

```java
String str;
str = new String("This is an example");
System.out.println("Initial string: \"" + str + "\"");
                           // Note the escaped quotes!
int l = str.length();
System.out.println("The length of the string is " + l);
char c = str.charAt(0);
System.out.println("The first character is " + c);
String str2 = str.substring(8,18);
System.out.println("The substring from char 8 to char 17 is \""
                   + str2 + "\"");
```

Your own complex types: classes

Class: group together variables that store your data.

```
class Person {
    String name;
    int age;
}
```

Your own complex types: classes

Class: group together variables that store your data.

```java
class Person {
    String name;
    int age;
}
```

Here: name and age are **fields** or **instance variables** of class Person.

Your own complex types: classes

Class: group together variables that store your data.

```java
class Person {
    String name;
    int age;
}
```

Here: name and age are **fields** or **instance variables** of class Person.

Using class Person:

```java
Person employee = new Person();
  // create instance or object of class Person
employee.name = "John Smith";
employee.age = 45;
System.out.println("BOSS:  How old are you, " + employee.name + "?");
System.out.println("EMPLOYEE:  I am " + employee.age
                                      + " years old.");
```

Boxed types

Simple types have “companion” complex types called **boxed types**:

Simple	Complex
int	Integer
double	Double
char	Character

Boxed types

Simple types have “companion” complex types called **boxed types**:

Simple	Complex
int	Integer
double	Double
char	Character

Benefits:

- Boxed types have methods like `Integer.parseInt()`.
- In some contexts Java expects a complex types.

Boxed types

Simple types have “companion” complex types called **boxed types**:

Simple	Complex
int	Integer
double	Double
char	Character

Benefits:

- Boxed types have methods like `Integer.parseInt()`.
- In some contexts Java expects a complex types.

Downside: efficiency.

Boxed types

Simple types have "companion" complex types called **boxed types**:

Simple	Complex
int	Integer
double	Double
char	Character

Benefits:

- Boxed types have methods like `Integer.parseInt()`.
- In some contexts Java expects a complex types.

Downside: efficiency.

Modern Java: use simple types and boxed types interchangeably.

```java
Integer i = 8;
int j = i + 1;
```

Branching: **if** ... **else**

```
java.util.Scanner scan = new java.util.Scanner(System.in);
System.out.println("Please choose an option:");
System.out.println("For 'Checking you balance', please enter 1");
System.out.println("For 'Purchases', please enter 2");
System.out.println("For any other query, please enter 0");
int choice = scan.nextInt();
if (choice == 1) {
    // go and check balance
} else if (choice == 2) {
    // go and purchase something
} else {
    // go and talk with a human operator
}
```

Branching: **switch** … **case** … **default**

```java
java.util.Scanner scan = new java.util.Scanner(System.in);
System.out.println(”Please choose an option:”);
System.out.println(”For 'Checking you balance', please enter 1”);
System.out.println(”For 'Purchases', please enter 2”);
System.out.println(”For any other query, please enter 0”);
int choice = scan.nextInt();
switch (choice) {
case 1:
    // go and check balance
    break;
case 2:
    // go and purchase something
    break;
default:
    // go and talk with a human operator
    break;
}
```

Enumerated types

List of “tags” as new complex data type:

```
enum Day {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY,
}
```

Use:

```
Day dayOfWeek = Day.MONDAY;
```

```
switch (day) {
case MONDAY: // not: 1 ("magic number")
    // do something here for this day
    break;
case WEDNESDAY: // not: 3
    // do something here for this day
    break;
...
}
```

The ternary/conditional operator

```java
java.util.Scanner scan = new java.util.Scanner(System.in);
System.out.println("Enter a number: ");
int i = scan.nextInt();
String s;
if (i > 5) {
    s = "Greater than 5";
} else {
    s = "Not greater than 5";
}
System.out.println(s);
```

vs

```java
java.util.Scanner scan = new java.util.Scanner(System.in);
System.out.println("Enter a number: ");
int i = scan.nextInt();
String s = (i > 5) ? "Greater than 5" : "Not greater than 5";
System.out.println(s);
```

Topics on Day 3

- Static vs dynamic typing
- Simple data types
- Complex data types: classes, enums
- Strings, boxed types
- Branching: `switch`, the ternary operator