VARIABLES AND TYPES

CITS1001
Scope of this lecture

• Types in Java
  • the eight primitive types
  • the unlimited number of object types

• Values and References

• The Golden Rule
Primitive types

- Every piece of data that is stored or moved around in Java must have a type
- Java supplies eight primitive types for the most fundamental pieces of data

short
int
long
float
double
byte

boolean
char
Integer primitive types

- Four of the types are used for storing whole numbers
- Each occupies a fixed amount of memory
  - byte uses 1 byte and stores integers from $-128$ to $127$
  - short uses 2 bytes and stores integers from $-32768$ to $32767$
  - int uses 4 bytes and stores integers from $-2^{15}$ to $2^{15} - 1$
  - long uses 8 bytes and stores integers from $-9223372036854775808$ to $9223372036854775807$
    (that is $-2^{63}$ to $2^{63} - 1$)
Floating point primitive types

- There are two types used for floating point values – in other words, numbers with decimal points
  - `float` uses 4 bytes and stores floating point numbers, positive or negative, in the range $1.4e^{-45}$ to $3.4028235e38$
  - `double` uses 8 bytes and stores floating point numbers, positive or negative, in the range $4.9e^{-324}$ to $1.7976931348623157e308$
Declaring variables

- In Java, all variables must be declared before they are used.
- The declaration specifies the type and name of the variable.
- When a primitive variable is created, a fixed amount of memory is set aside for that variable.
  - `byte b;`
  - `double temp;`
  - `int balance;`
Assignments to numeric types

• An assignment sets the value of a variable
  int x;       // this is a declaration
  x = 100;     // this is an assignment

• Here are some legal assignments
  int x;
  double y;
  double z;

  x = 200;
  y = -354.278;
  z = 23e5; // this is scientific notation
Typecasting

• The compiler will warn you if you attempt to use a “smaller-sized” variable than it thinks the value needs.

• For example, if you attempt

```c
float b;
b = 12.35;
```

then the compiler will warn you of a “possible loss of precision” because 12.35 is regarded as a `double`.

• If this is `really` what you want, then you can `cast` the value

```c
float b;
b = (float) 12.35;
```

• for `literals`, there is a “quick cast” – `b = 12.35f;`
Non-numeric primitive types

• **boolean** is used to hold one of the two boolean values **true** or **false**

  ```java
  boolean isDone;   // declaration
  isDone = false;   // assignment
  ```

• **char** uses two bytes and holds a Unicode **character**

  ```java
  char ch;           // declaration
  ch = ‘A’;          // assignment
  ```
Literals

- Actual values that you type into the source code are called *literals* - these are things like

  100, -23, true, 'Z', 37, 1e-5, 22, 3.1415926

- *Everything* in Java has a type, even the literals, so the types of these are respectively

  int, int, boolean, char, int, double, int, double
Initialization

- If a variable is given an initial value directly after it is declared, then you can combine the two steps into one

  ```java
  int x;
  x = 10;
  boolean isFinished;
  isFinished = true;
  ```

  is equivalent to

  ```java
  int x = 10;
  boolean isFinished = true;
  ```
Type-checking

- The Java compiler does extensive type-checking to make sure that the program does not have any obvious problems.
- The compiler will not allow the wrong type of value to be assigned to a variable:
  ```java
  int x;
  x = true;   // cannot assign boolean to int
  ```
- Similarly, the compiler will not allow the wrong type of value to be used as a method or constructor argument:
- What about this?
  ```java
  int x = 20;
  double y;
  y = x;
  ```

  Although the types don’t match perfectly, the computer knows that there is a value of type `double` corresponding to any value of type `int` – in this case the value `20.0` is assigned to `y`.
Variable declarations

- *Declaring* reference types is just the same as for primitive types, with the type occurring before the name
  
  ```java
  String s;
  BankAccount b1;
  BankAccount b2;
  SimpleCanvas sc;
  ```

- This has created the variables – the “names” for the objects – but we have not actually created the objects themselves

Creating objects is (almost always) done with the Java keyword `new`
Creating objects

• To actually create the object, the constructor is called
  
  \[
  \text{sc} = \text{new SimpleCanvas();}
  \]
  
  \[
  \text{bl} = \text{new BankAccount(“BillG”, 12345, 10000);}
  \]

• An \textit{object} is fundamentally different from a primitive value
  
  • It does not have a fixed size when it is created
  
  • Its size can change during runtime

• For this reason, objects are stored in a different part of the computer’s memory, and the value that is stored in the variable is a \textit{reference} to the object
  
  • A reference contains the information needed to \textit{find} the actual object in the computer’s memory
  
  • The variable \textit{refers to} an object (rather than \textit{contains} an object)
The variable b1 contains a reference to some location in the heap.

The actual object that b1 refers to is stored on the heap.

Some other object.
The Golden Rule

- The behaviour of reference types and primitive types appears quite different, but it all hinges on the following observation
  - A variable of primitive type *contains* a value
  - A variable of reference type *refers* to an object

*The Golden Rule*

*Whenever* a variable is used, it is the *contents* of the shoebox that is used
  - this is the *value* for a variable of primitive type
  - this is the *reference* for a variable of reference type
Summary

- Java supplies eight *primitive* types for the most fundamental pieces of data.
- A variable name and type must be *declared* before the variable can be used in a program.
- The Java compiler does extensive type-checking to make sure that the program does not have any *obvious* problems.
- Creating objects is (almost always) done with the Java keyword `new`.
- Every variable has a *name* used to refer to it.
- Every variable has a *value*, that changes during program run time.