Data Types and Variables

• Variables are names for places in memory in which to put values
• Data types describe the nature of the variable
• Data types limit the operations on each value
• Variables bind data types to values in memory
• They reduce erroneous programming, enhance portability and discipline your code.
• Any value of a data type is actually an object
Declaring Variables

• To declare variables one needs to specify the datatype and the variable name.

• Common datatypes are:
  – Boolean/bool
  – Character/char
  – Short/short
  – Integer/int
  – Long/long
  – Float/float
  – Double/double
  – String/String

• Instead of calling them datatypes, we will call these class-types. These behave like data-types yet with added functionality.
Identifiers

• Each variable or other named program component requires a unique name, or identifier.

• Identifiers:
  – start with a $, an underscore, or an alphabetic character
  – followed by one or more alphanumeric, $ or underscore characters
Identifiers (cont.)

• The following are legal identifiers:
  – Dummy
  – box$
  – A37
  – post_code

• The following are not legal identifiers:
  – rain@noon
  – 2by4
  – post_code

• Case is significant:
  – identifiers Abc, aBc and abC are different
Declaring Variables

• The following statements creates a variable of type Integer:
  – Integer i; (Java)
  – int i; (C#)

• Since Integer/int is a class-type, then i is a class-type variable

• In Java/C# ‘i’ is not a real integer, it is just a variable that can hold an integer.

• To create an integer we need to allocate space for the integer and also give the integer an initial value

• This is done by using the new constructor in Java. In C# this is sadly hidden.
Creating an Object

• *new* is an operator that
  – allocates new space depending on the class-type
  – returns a reference to the space
  – Example: \( i = \text{new Integer(2)}; \)
  – Every time *new* is called a new object is created

• In C# whenever we have
  – int \( i=2; \)
    is translated in the background to
  – int \( i = \text{new int(2)}; \)
  The end result will be the following:

  \( \textbf{Object of Integer Type} \)

  \[
  \begin{array}{c}
  \text{i} \\
  \rightarrow \\
  2
  \end{array}
  \]

  – The *new* operator creates an \textbf{object} of type
    Integer
  – Note that \( i \) is just a reference to (points to) the
    object and is not the object itself
  – Every object needs to be given an appropriate
    initial value
Variables

One can assign new objects to variables of the same type using the assign operator ‘=‘:

- Assignment does not modify the value of the object
- Several variables can point to the same object

- When an object is not pointed to by any variable, it is deleted automatically.
  - This process is known as garbage collection

- Two variables can be compared using == or !=.
  - == returns true only if both variables point to the same object (regardless of contents)
null

- Any variable may be given the value `null`.
- This means that this variable points to nothing.
- All variables are given this default value until assignment.
Object Access

• Object of the mentioned class types can be created using the new operator

• To access the object, we make use of method calls

• Methods are like function calls that return a specific value

• To access a method, one uses the class type variable followed by the method name separated by the ‘.’ character

  Example: In Java: i.intValue(), in C#: i gives the value stored in the object pointed to by i.
Useful Methods

• Objects allow only legal operations to be performed on them by offering methods.
• These legal operations are dictated by the class type
  – Compare to data-types
• Common methods
  – Integer: intValue()/ not defined
  – Character: charValue()/ not defined
  – Float: floatValue()/ not defined
  – Double: doubleValue()/ not defined
  – Long: longValue()/ not defined
  – Boolean: booleanValue()/ not defined
  – Short: shortValue()/ not defined
Useful Methods (cont.)

- In addition, each of the class-types that we discussed have a method called `toString() / ToString()`. This will give the string representation of any object.
- In Java, each of these class types will not allow you to change the contents of the object.
- In C# you still cannot really change contents yet is hidden away by compiler
- To manipulate values you will need to create a new object
- For example in Java to increment a variable the following can be used
  - `Integer i;`
  - `i = new Integer(5);`
  - `i = new Integer(i.intValue()+1);`
- In C# its simply `i++;`
Exercises

• Create several objects containing the same value.
• Compare them using the == operator to see if the objects are alike
• Now compare their contents
• Check the documentation for other methods that a class-type has and try to use them.