CSE 1020: Unit 1

Topic: Abstraction

To do: Textbook Chapter 1; Lab 1

Outline

- **Abstraction**
- **Hardware abstraction**
- **Software abstraction**
- **Data abstraction**

Outline

- **Abstraction**
- **Hardware abstraction**
- **Software abstraction**
- **Data abstraction**

3

Abstraction

What is abstraction

- An abstraction is a set of data and/or operations that is provided to some users.
- How the data/operations are implemented is hidden from the users.
- This process is referred to as information hiding or encapsulation.
- All the user knows is
  - How to invoke operations (names, parameters, etc.)
  - What the results and effects are

  We refer to this as the abstraction's Application Programming Interface (API).

4

Abstraction

Why abstraction is important to us

- A user can use the abstraction without knowing the details of the implementation.
- This concept is very important in the development of large software systems involving millions of lines of code.

5

Abstraction

Why abstraction is important to us

- A user can use the abstraction without knowing the details of the implementation.
- This concept is very important in the development of large software systems involving millions of lines of code.

Example

- In modeling the Toronto Stock Exchange we may choose to abstract the notion of a "stock".
- This allows others to operate on a stock (e.g., buy/sell) without being concerned with how we maintain a particular stock value.

6

Abstraction

Abstract data types

- An abstract data type (ADT) is…
- A set of *values* that belong to the data type, e.g.,
  - integers
  - strings
  - etc.
- A set of *operations* on these values, e.g.,
  - addition for integers
  - concatenation for strings.

7

Abstraction

Abstract data types

- Users of the ADT are told…
- How the operations can be invoked, e.g.,
  - name
  - parameters
- What the operation's preconditions are
  - What is required for the operation to be possible
  - E.g., for division the divisor must not be zero.
- What the operation's postconditions are
  - what effects it has
  - what results it returns
- This constitutes the API of the ADT.
- How the ADT is implemented is kept hidden from the users.

8

Abstraction

We need two fundamental kinds of abstraction

1. Abstractions that capture operations performed on data (procedures).
2. Abstractions that capture the values of items of interest (data).

Abstraction

A need

- Before writing a program to solve a problem or defining an operation to manipulate data,
- must have an procedure that can do the work.

Abstraction

Problem

- Determine how many months it takes to pay back a loan given the loan amount, monthly payment amount and interest rate.

11

Abstraction

Problem

- Determine how many months it takes to pay back a loan given the loan amount, monthly payment amount and interest rate.

Solution procedure

1. Initialize *monthsRequired* to 0.
2. Repeat (i), (ii) and (iii) while *amountOwed* > 0.
   - (i) Add *monthlyInterest* to *amountOwed*.
   - (ii) Subtract *monthlyPayment* from *amountOwed*.
   - (iii) Increment *monthsRequired* by 1.
3. Report *monthsRequired* as the answer.

12

Abstraction

A need

- Before writing a program to solve a problem or defining an operation to manipulate data,
- must have an procedure that can do the work.

A good solution method should be

- *Unambiguous*: Leaves no doubt about what operation to perform at each step.
- *Executable*: Performable on the computer.
- *Terminating*: Guaranteed to come to an end.
- We refer to a method with these properties as an algorithm.

13

Abstraction

Another need

- Programs (as well as algorithms and ADTs) use variables to maintain values.
- For example
  - monthsRequired
  - amountOwed
  - Etc.

  in the loan example.
- A variable has a particular value at a given time, and it changes as a program executes.

14

Abstraction

We have introduced

- Abstraction as a key notion in computer science.
- We abstract both
  - data
  - procedures (for manipulating data)
- Some have referred to computer science as the science of abstraction.

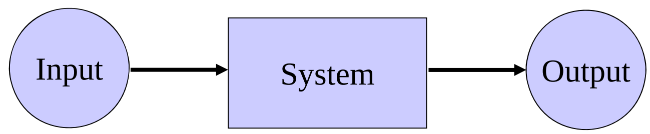
15

Outline

- **Abstraction**
- **Hardware abstraction**
- **Software abstraction**
- **Data abstraction**

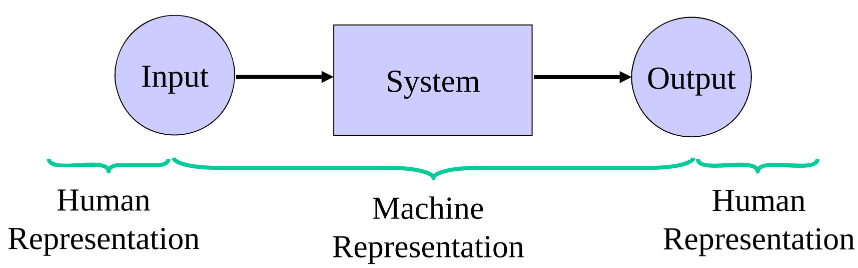
16

A simple hardware model



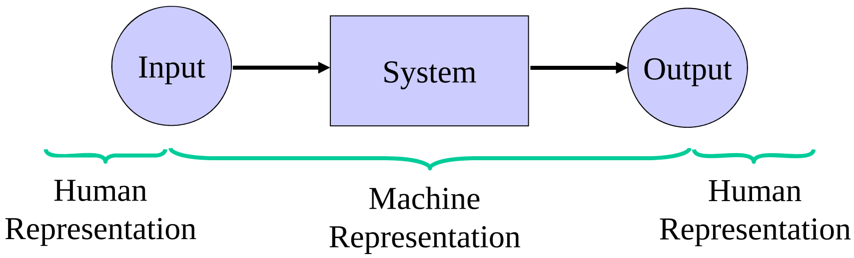
17

A simple hardware model



18

A simple hardware model

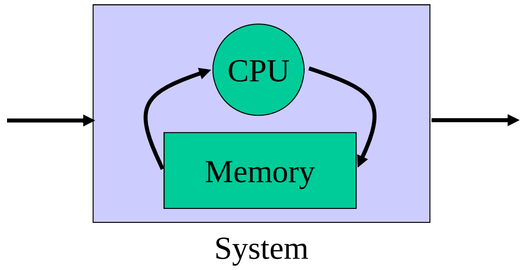


Remarks

- Input/output (I/O) units are translators.
  – Keyboard, mouse, microphones…
  – Screen, speaker…
- This model is so general as to work for almost anything.

19

A simple hardware model



Central Processing Unit (CPU)

- Performs arithmetic and logic operations
- Keeps track of next instruction

(Main) Memory (RAM & ROM)

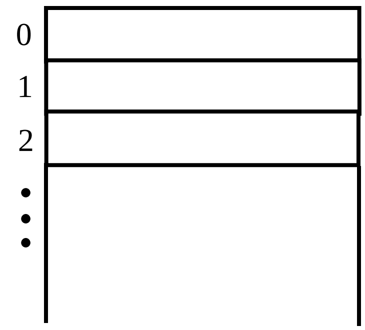
- Stores data
- Stores programs

20

A simple hardware model

Memory

- A set of cells
- Each with an address
- Each with contents

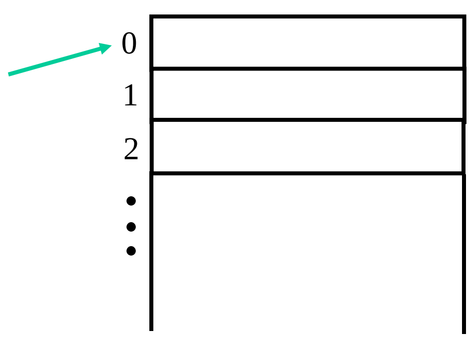


21

A simple hardware model

Memory

- A set of cells
- Each with an address
- Each with contents

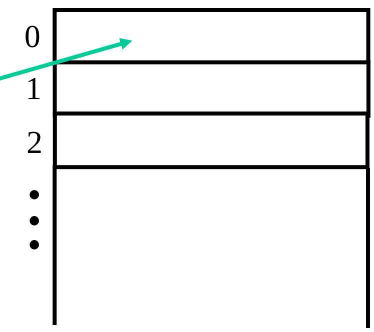


22

A simple hardware model

Memory

- A set of cells
- Each with an address
- Each with contents



23

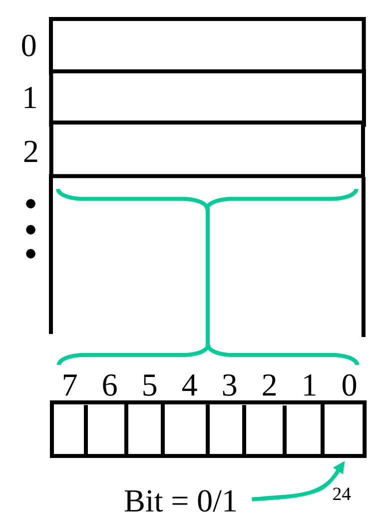
A simple hardware model

Memory

- A set of cells
- Each with an address
- Each with contents

Cell

- Contains only one element at a time
- Capacity of 1 byte (8 bits)



24

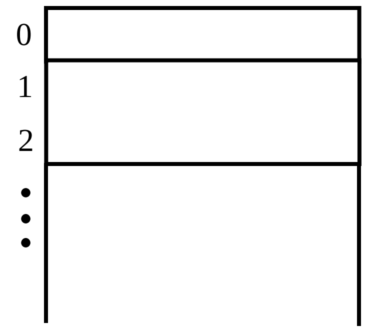
A simple hardware model

Memory

- A set of cells
- Each with an address
- Each with contents

Expand cells

- Concatenate bytes
- Address that of lowest byte in group
- Come in "byte size" packages.



25

A few more hardware concepts

Secondary storage

- Hard disk, floppy disc, CD-ROM, tape, etc. for data storage.
- Slower access and larger capacity than main memory.
- Persistent.

Bus

- A thick set of wires.
- Allows data to be moved between CPU, memory and other components.

Network connection

- Allows individual computers to communicate with other computers and shared peripheral devices.

26

Storage management

Storage considerations

- Keeping track of memory cells and their contents.
- Keeping track of all symbolic names and their values.

27

Storage management

Storage considerations

- Keeping track of memory cells and their contents.
- Keeping track of all symbolic names and their values.

Multilayer abstraction

- Memory manager
- Symbol manager
- Benefits: Changes at one level of implementation need not impact other levels of implementation.

28

Storage management

Storage considerations

- Keeping track of memory cells and their contents.
- Keeping track of all symbolic names and their values.

Memory manager

- Responsibilities:
  - Keeps track of memory cells and contents.
- Tasks:
  - Allocate/deallocate
  - Read/write
- Implementation:
  - At level of operating system.

29

Storage management

Storage considerations

- Keeping track of memory cells and their contents.
- Keeping track of all symbolic names and their values.

Symbol manager

- Responsibilities:
  - Keeps track of names and values.
- Tasks:
  - Declare
  - Evaluate
  - Assign
- Implementation:
  - At level of compiler

30

Storage management

Storage considerations

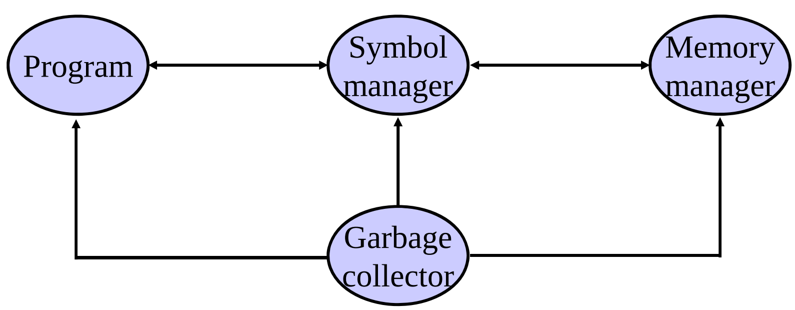
- Keeping track of memory cells and their contents.
- Keeping track of all symbolic names and their values.

Garbage collection

- In certain programming languages (e.g., Java) memory cells that are no longer in use are automatically recycled for reuse.
- We refer to this process as garbage collection.

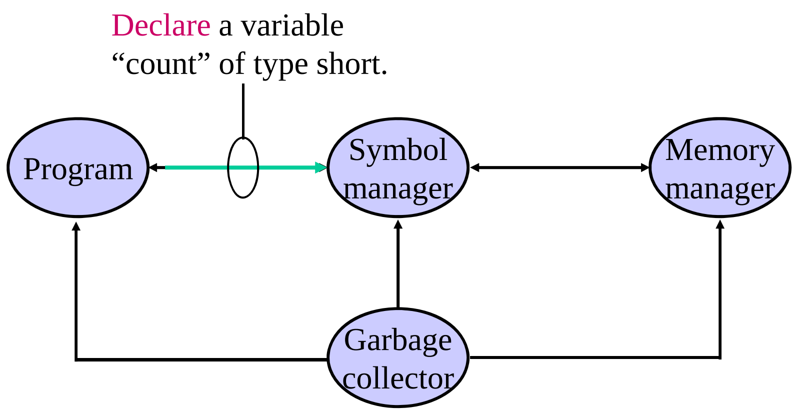
Storage management

Pictorial representation



Storage management

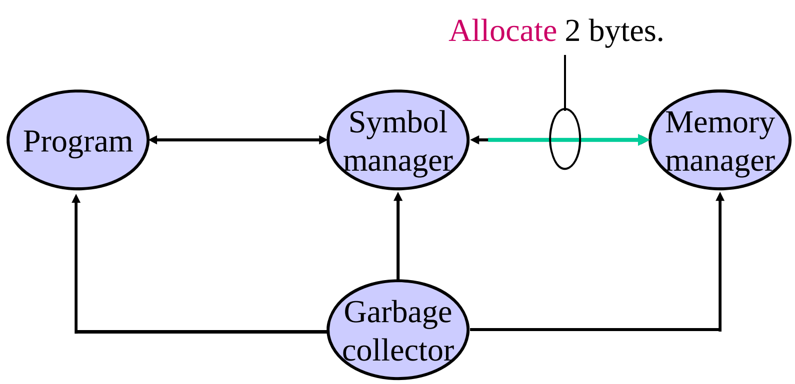
Pictorial representation



33

Storage management

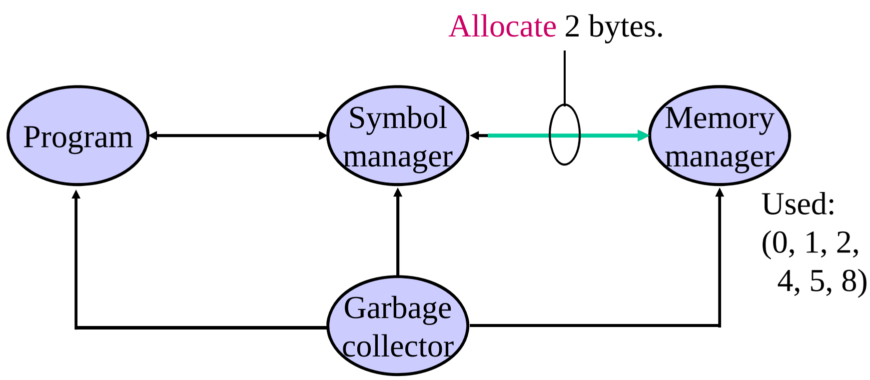
Pictorial representation



34

Storage management

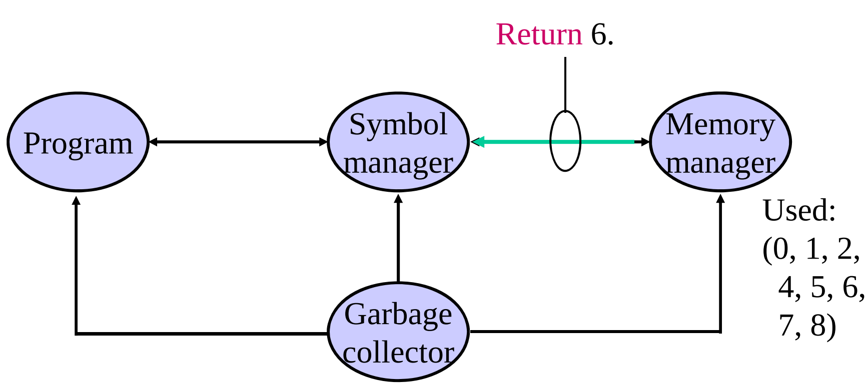
Pictorial representation



35

Storage management

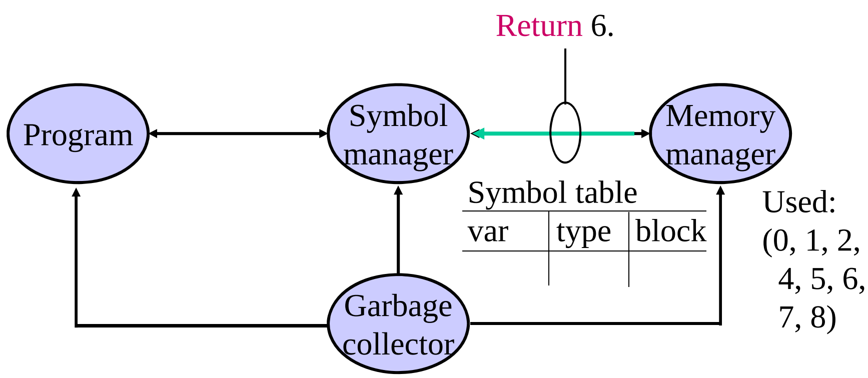
Pictorial representation



36

Storage management

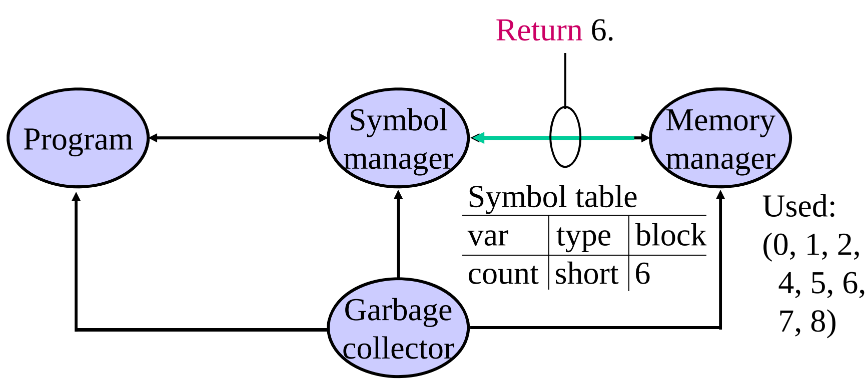
Pictorial representation



37

Storage management

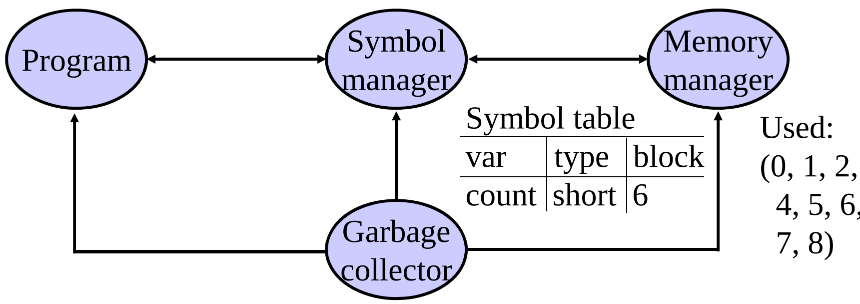
Pictorial representation



38

Storage management

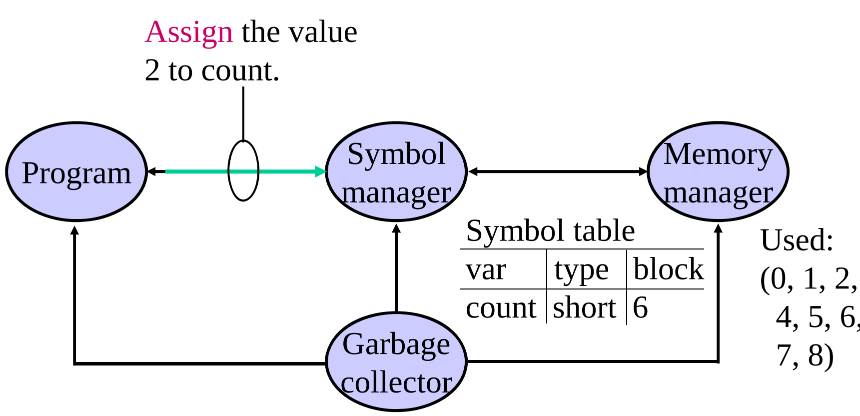
Pictorial representation



39

Storage management

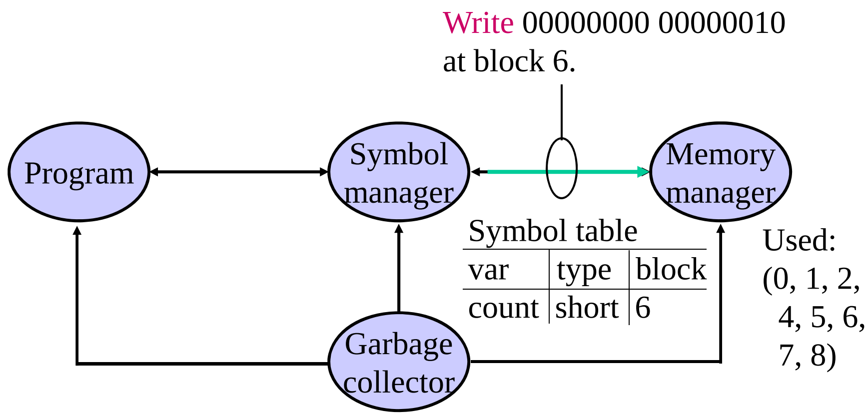
Pictorial representation



40

Storage management

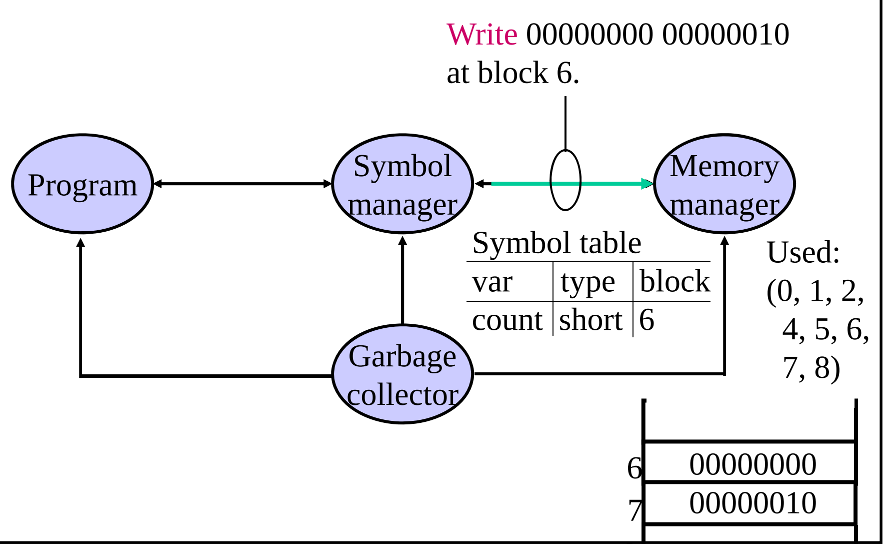
Pictorial representation



41

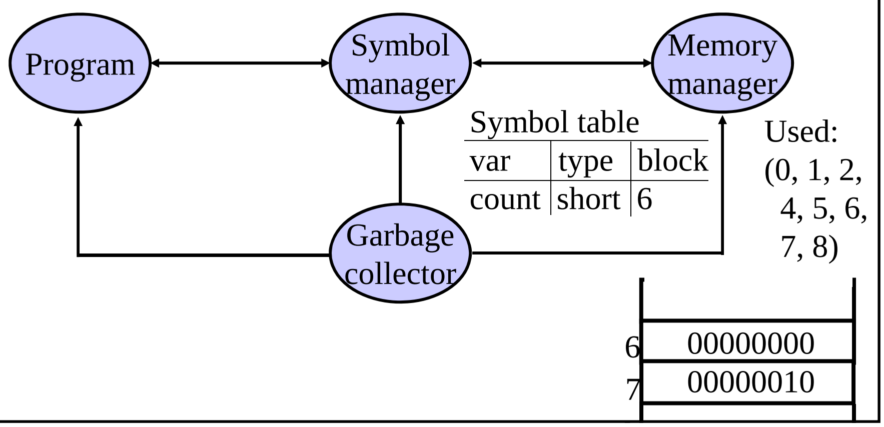
Storage management

Pictorial representation



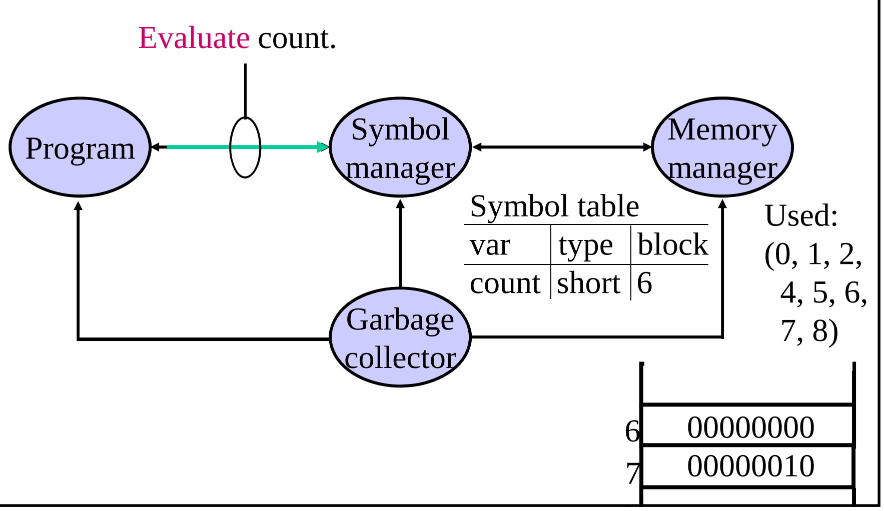
Storage management

Pictorial representation



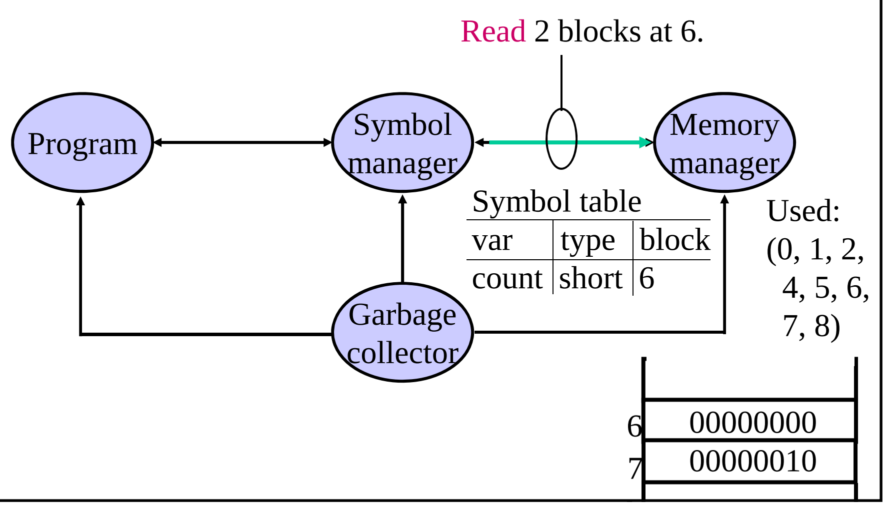
Storage management

Pictorial representation



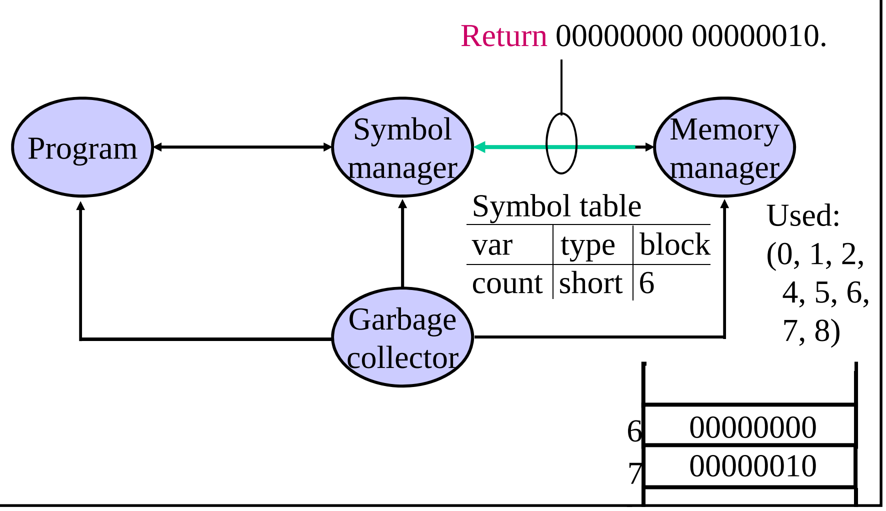
Storage management

Pictorial representation



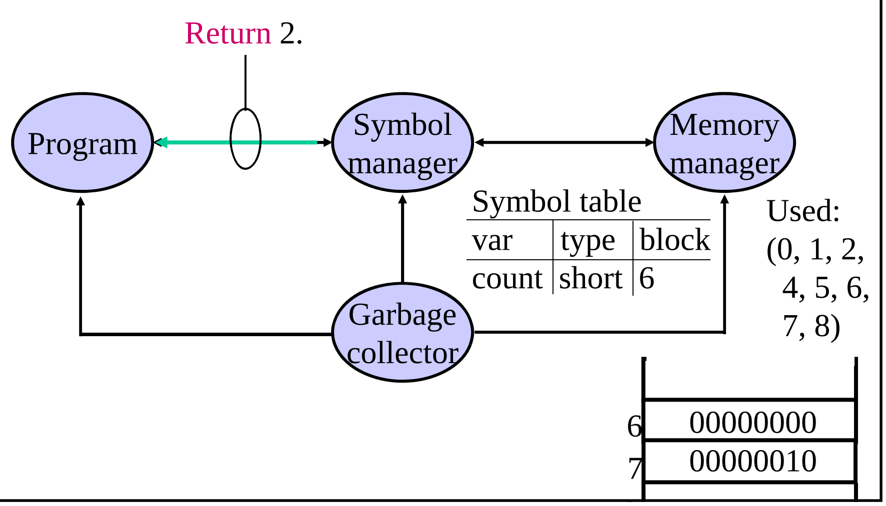
Storage management

Pictorial representation



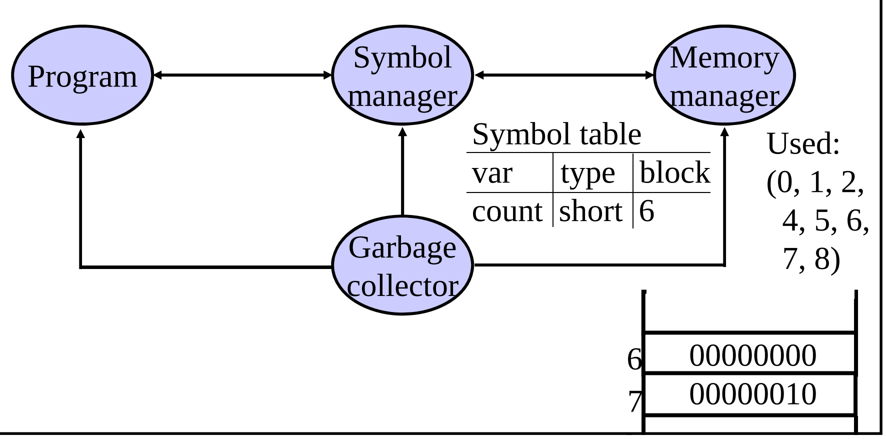
Storage management

Pictorial representation



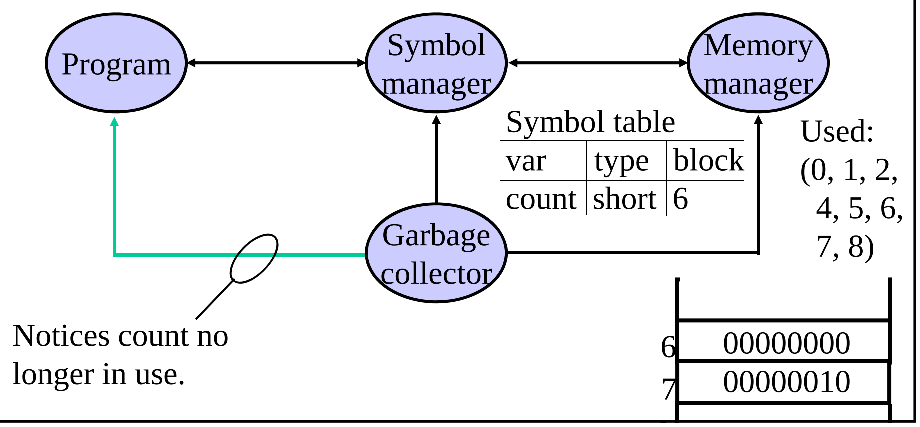
Storage management

Pictorial representation



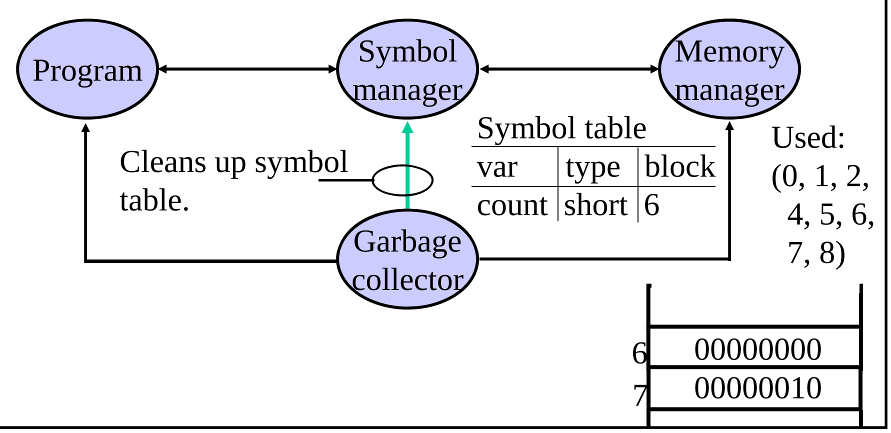
Storage management

Pictorial representation



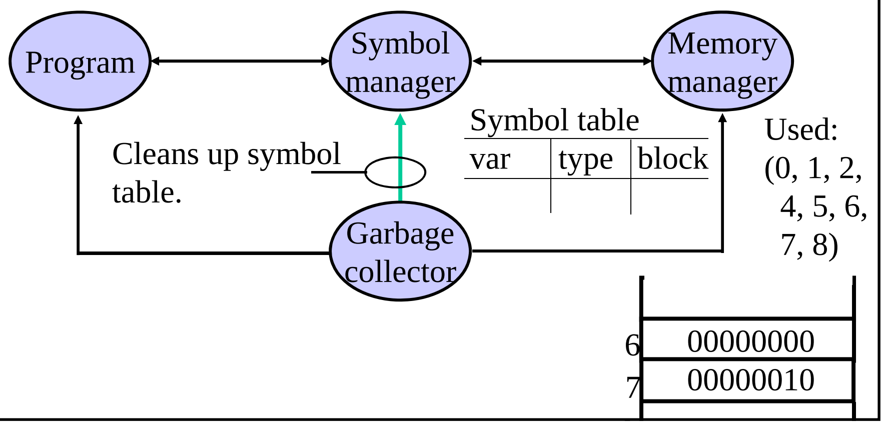
Storage management

Pictorial representation



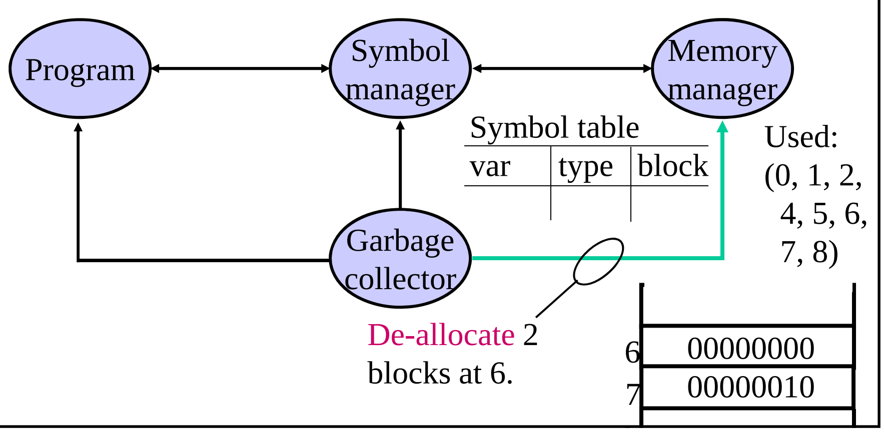
Storage management

Pictorial representation



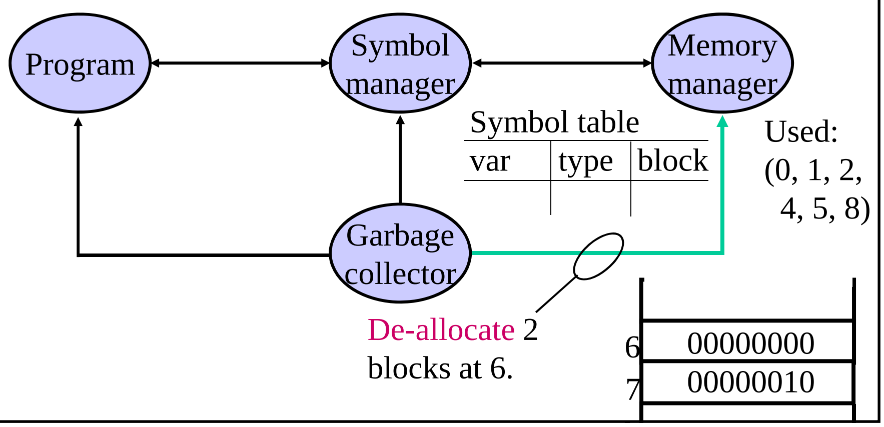
Storage management

Pictorial representation



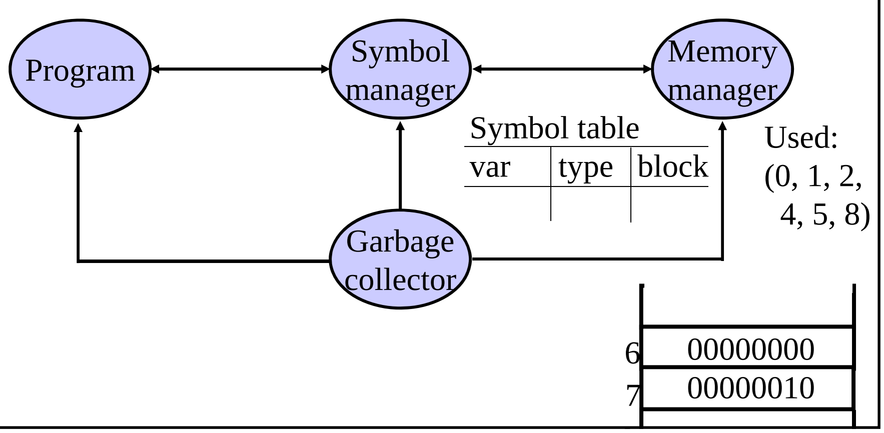
Storage management

Pictorial representation

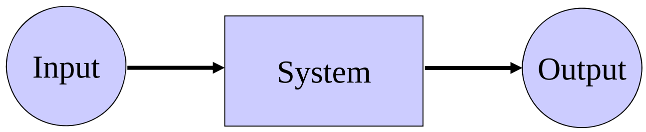


Storage management

Pictorial representation

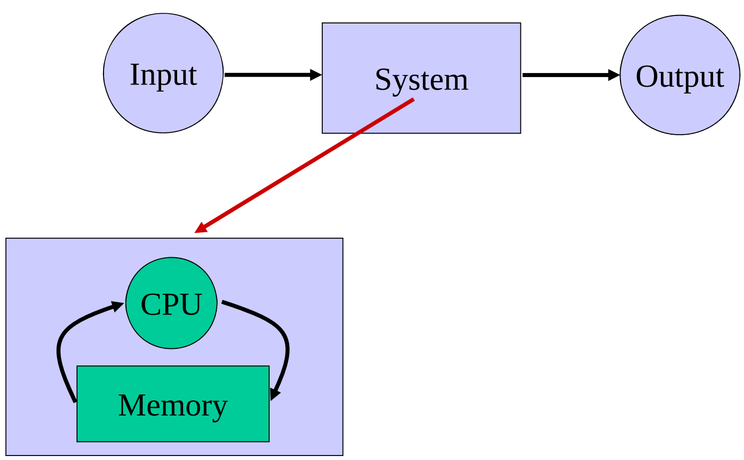


Our hardware abstraction



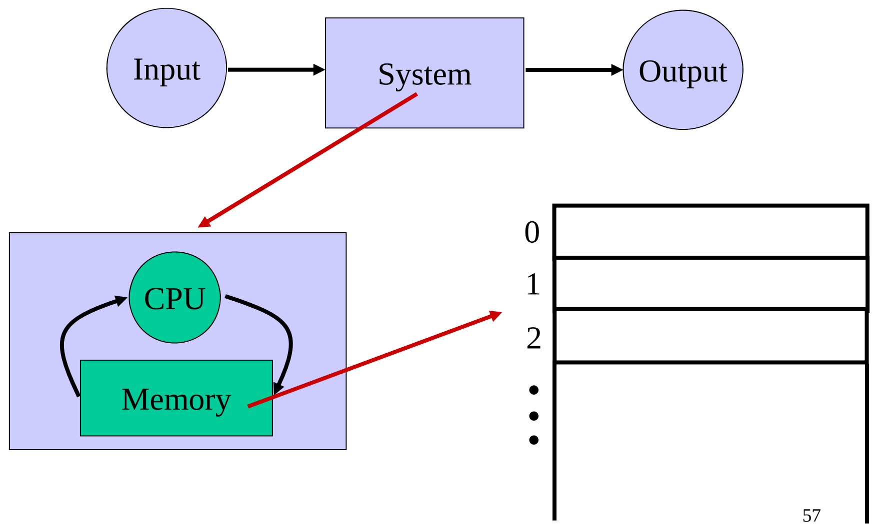
55

Our hardware abstraction



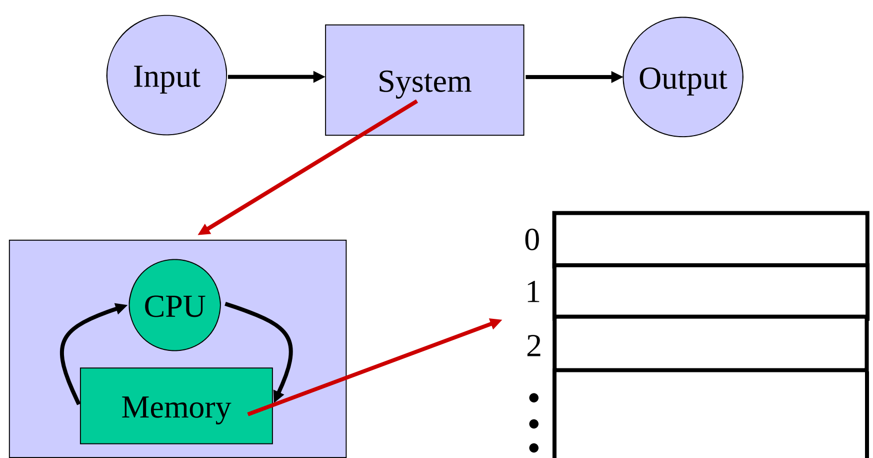
56

Our hardware abstraction



57

Our hardware abstraction



Storage management requires tracking memory and contents; names and values.

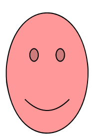
58

Outline

- Abstraction
- Hardware abstraction
- **Software abstraction**
- Data abstraction

59

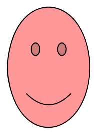
Software abstraction



How can we bridge this (wide) gap?

60

Software abstraction



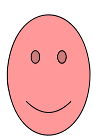
How can we bridge this (wide) gap?

Consider the human language sentence

- Time flies like an arrow.

61

Software abstraction



How can we bridge this (wide) gap?

Human language
- ambiguous
- context disambiguates

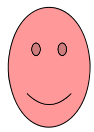
Machine language
- requires unambiguous specifications
- context free

Consider the human language sentence

- Time flies like an arrow.

62

Software abstraction



How can we bridge this (wide) gap?

Human language
- ambiguous
- context disambiguates

Machine language
- requires unambiguous specifications
- context free

Conclusion
- One side or the other must give in.
- The machine cannot accommodate.

63

Software abstraction

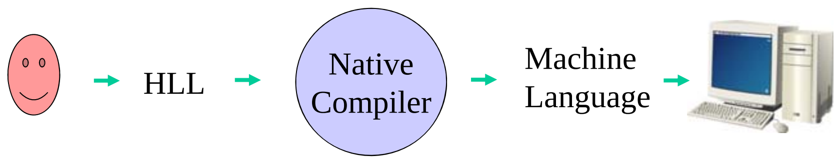
Build a bridge to the human
- Introduce high-level programming languages
- Abstract from the machine primitives
  - Memory address → variable names
  - Machine instructions → methods, procedures, functions (algorithms)
- Easier for humans to work with
  - Documentation critical
- But still context free
  - Require extreme precision of thought

64

Software abstraction

Build a bridge to the machine

- High-level language (HLL) must be translated to machine language
- Two approaches
  1. Native compiler
     + job done once and for all
     - final product platform specific

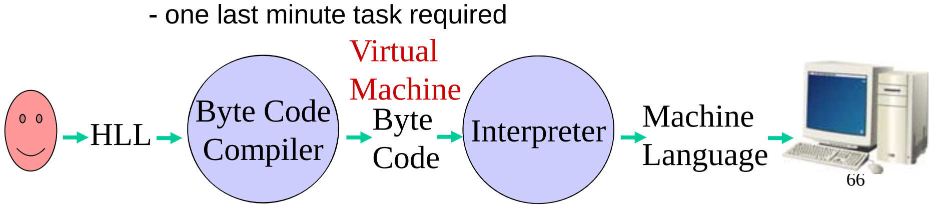


65

Software abstraction

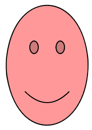
Build a bridge to the machine

- High-level language (HLL) must be translated to machine language
- Two approaches
  1. Native compiler
     + job done once and for all
     - final product platform specific
  2. Byte code compiler
     + job (almost) done and platform independent
     - one last minute task required



66

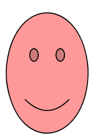
Software abstraction



Provides the
bridge across
this gap.

67

Software abstraction



Provides the
bridge across
this gap.

Comprised of

- Programs written in a computer language
- Associated documentation

Goal: Must be readily comprehended by

- Human → good style
- Machine (via compiler/interpreter) → unambiguous

68

A first program

3 steps in implementation

1. Edit
2. Compile
3. Run

69

A first program

3 steps in implementation

→ 1. Edit
2. Compile
3. Run

70

A first program

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

71

A first program

Edit

- In an editor we enter

Includes useful stuff (classes) in the type package.

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

72

A first program

Edit

- In an editor we enter

A blank line to separate logically separate pieces of code.

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println(“Hello, world!”);
     }
}
```

- and save to a file Hello.java

73

A first program

Starts a new class, Hello

- A class is a way to group together related data and operations.

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println(“Hello, world!”);
     }
}
```

- and save to a file Hello.java

74

A first program

Remarks

- public denotes that this class is available to others.
- class and file names must be consistent.

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

75

A first program

Remarks

- public denotes that this class is available to others.
- class and file names must be consistent.

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

76

A first program

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

Defines

- A "method" called main.
- A method is a set of instructions for carrying out a task.
- Methods must be in classes.

77

A first program

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

The term static specifies that the main method neither inspects nor modifies customized Hello class copies (objects).

78

A first program

The term void specifies that the main method does not yield a return value.

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

79

A first program

The name of the main method.

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

80

A first program

Specifies the (command line) arguments for the main method.

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

81

A first program

Invokes an (imported) method to print the string "Hello, world!", e.g., on the screen.
• println is a method in the class IO that takes a string argument.

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

82

A first program

Delimiters

• We use curly brackets, { }, to delimit portions of our code

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

83

A first program

Delimiters

• We use curly, { }, brackets to delimit portions of our code
• These curly brackets mark the start/end of the class Hello

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

84

A first program

Delimiters

- We use curly, { }, brackets to delimit portions of our code
- These curly brackets mark the start/end of the method main

Edit

- In an editor we enter

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

85

A first program

3 steps in implementation

1. Edit
2. Compile
3. Run

86

A first program

Compile

- At our command line prompt…
- we invoke the compiler…
- to produce byte code to be interpreted by computer.

% javac Hello.java

87

A first program

3 steps in implementation

1. Edit
2. Compile
3. Run

88

A first program

Run

- We now convert the byte code produced by the compiler…
- … to native code that executes on the machine at hand.
- At the command line prompt we invoke the interpreter

```
% java Hello
```

Success

- Will produce on the screen

```
Hello, world!
```

89

A first program

More to be done

- What have we forgotten?

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

90

A first program

Documentation

```
/*
Java program to print a greeting. Upon invocation it
prints "Hello, world!" to standard out.

Author: Richard Wildes                    Date: 05/05/13
*/

import type.lang.*; // import type package for general utils.

// Definition of the Hello class.
public class Hello
{   public static void main(String[ ] args)
    {    // Print to standard out.
         IO.println("Hello, world!");
    }
}
```

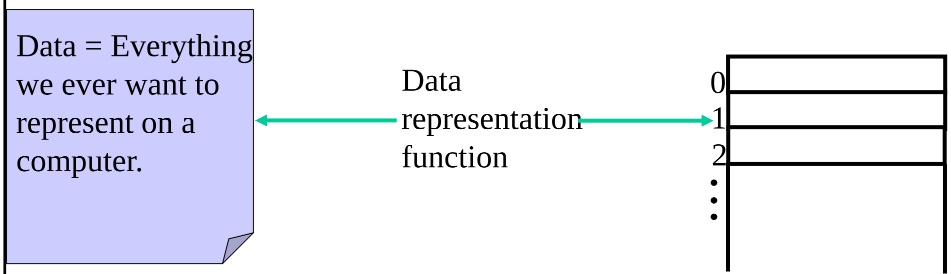
91

Outline

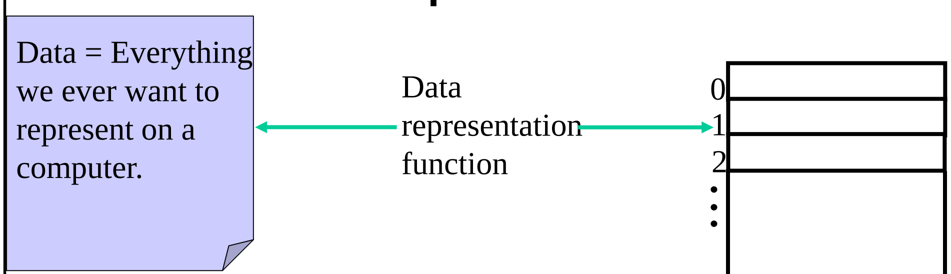
- Abstraction
- Hardware abstraction
- Software abstraction
- **Data abstraction**

92

Data representation



Data representation



Remark

- With ingenuity wide variety of data can be so mapped.
- We restrict our attention to
  - numbers
  - characters
  - booleans

Data representation

Numbers
- integers
- reals

Characters
- letters
- digits
- symbols ( ) + -, etc.

Booleans
- true
- false

95

Data representation

Numbers
- integers
- reals

Characters
- letters
- digits
- symbols ( ) + -, etc.

Booleans
- true
- false

Question: Why have both digits And integers?
Answer: Numbers support arithmetic operations; characters support disection operations. Representation depends on what you want to do.

96

Data representation

Numbers
- integers
- reals

Characters
- letters
- digits
- symbols ( ) + -, etc.

Booleans
- true
- false

97

Data representation

Integers
- Map to memory under binary representation.
- According to anticipated usage, allocate different amounts of memory

98

Data representation

Integers

- Map to memory under binary representation.
- According to anticipated usage, allocate different amounts of memory

Examples (unsigned)

Base 10

255 = 2x100 + 5x10 + 5x1
= 2x10^2 + 5x10^1 + 5x10^0

99

Data representation

Integers

- Map to memory under binary representation.
- According to anticipated usage, allocate different amounts of memory

Examples (unsigned)

Base 10

255 = 2x10^2 + 5x10^1 + 5x10^0

Base 2

11111111 = 1x2^7 + 1x2^6 + 1x2^5 + 1x2^4 + 1x2^3
+ 1x2^2 + 1x2^1 + 1x2^0

100

Data representation

Integers

- Map to memory under binary representation.
- According to anticipated usage, allocate different amounts of memory

Examples (unsigned)

Base 10

0 = 0x10^0

Base 2

0 = 0x2^0

101

Data representation

Integers

- Map to memory under binary representation.
- According to anticipated usage, allocate different amounts of memory

Examples (unsigned)

Base 10

1 = 1x10^0

Base 2

1 = 1x2^0

102

Data representation

Integers

- Map to memory under binary representation.
- According to anticipated usage, allocate different amounts of memory

Examples (unsigned)

Base 10

2 = 2x10^0

Base 2

10 = 1x2^1 + 0x2^0

103

Data representation

Integers

- Map to memory under binary representation.
- According to anticipated usage, allocate different amounts of memory

Examples (unsigned)

Base 10

3 = 3x10^0

Base 2

11 = 1x2^1 + 1x2^0

104

Data representation

Integers

- Map to memory under binary representation.
- According to anticipated usage, allocate different amounts of memory

Examples (unsigned)

Base 10

32 = 3x10^1 + 2x10^0

Base 2

100000 = 1x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 0x2^0

105

Data representation

Integers

- Map to memory under binary representation.
- According to anticipated usage, allocate different amounts of memory

Examples (unsigned)

Base 10

255 = 2x10^2 + 5x10^1 + 5x10^0

Base 2

11111111 = 1x2^7 + 1x2^6 + 1x2^5 + 1x2^4 + 1x2^3 + 1x2^2 + 1x2^1 + 1x2^0

106

Data representation

Integers

- Map to memory under binary representation.
- According to anticipated usage, allocate different amounts of memory

Encompassing negative integers

- We also need to include negative integers
- With, e.g., 8 bits we are restricted to 256 total values
- Approach
  - – Use left most bit for sign (e.g., 0 pos; 1 neg)
  - – Use (approx.) half the total values for positives.
  - – Use (approx.) half the total values for negatives.
  - – Reserve a value for zero (e.g., 00000000)

107

Data representation

Java integer types

bit	byte	KeyWord	range (approx.)
8	1	byte	+/- 127
16	2	short	+/- 32K
32	4	int	+/- 2G
64	8	long	big (+/- 9 quintillion)

Nomenclature

$$K \equiv 2^{10} = 1024 \approx 1{,}000$$

$$M \equiv 2^{20} = K * K \approx 1{,}000{,}000$$

$$G \equiv 2^{30} = M * K \approx 1{,}000{,}000{,}000$$

108

Data representation

Java integer types

bit	byte	KeyWord	range (approx.)
8	1	byte	+/- 127
16	2	short	+/- 32K
32	4	int	+/- 2G
64	8	long	big

Remarks

- Why not always choose the largest size?
  - Incurs increased processing time due to overhead in dealing with larger memory chunks.
- In 1020: Use int as default.

109

Data representation

Declaration

- In our computer programs, we will want to provide symbolic names for particular instances of a data type.
- We do this via declaration.
- In Java we declare an int via

```
int   qty;
```

110

Data representation

Declaration

- In our computer programs, we will want to provide symbolic names for particular instances of a data type.
- We do this via declaration.
- In Java we declare an int via

int qty;

Data type: Tells Java what type of item this is.

Whitespace: Tells Java That previous item has Ended. Free form use legal; But choose for readability.

Name: Tells Java the symbol to associate with this declaration. Choose for style.

Semicolon: Tells Java The statement is over.

111

Data representation

Declaration (behind the scenes)

- When the processor encounters our declaration

  int qty;

  - 4 bytes of free memory are allocated
  - The name qty is associated with the allocated memory location

Data representation

Declaration (behind the scenes)

- When the processor encounters our declaration

  int qty;

  - 4 bytes of free memory are allocated
  - The name qty is associated with the allocated memory location

Initialization

- In Java, declaration does not provide an initial value.
- To provide a value we use an assignment statement

  qty = 27;

- Can also combine declaration and initialization

  int qty = 27;

- In either case the value 27 has been stored in the memory location associated with qty

113

Data representation

Numbers

- integers
- reals

Characters

- letters
- digits
- symbols ( ) + -, etc.

Booleans

- true
- false

114

Data representation

Reals: The challenge

- It is theoretically impossible to represent real numbers on a digital computer.
- Certain reals (e.g., irrational numbers) cannot be captured with a finite representation.
- So we make a compromise between
  - Range represented
  - precision

115

Data representation

Reals: The IEEE 754 Standard

- Consider a number, say, 7412.3898
- Write it as .74123898x10^4
- Now we just need to represent 2 integers
  - 74123898
  - 4
- But still limited precision due to finite amount of memory available to represent 74123898
  - We speak of the number of significant figures as those that are captured under the representation

116

Data representation

Reals: The IEEE 754 Standard

- Consider a number, say, 7412.3898
- Write it as .74123898x10^4
- Now we just need to represent 2 integers
  - 74123898
  - 4
- But still limited precision due to finite amount of memory available to represent 74123898
  - We speak of the number of significant figures as those that are captured under the representation
  - 74123898→ 2 significant figures → 74

117

Data representation

Reals: The IEEE 754 Standard

- Consider a number, say, 7412.3898
- Write it as .74123898x10^4
- Now we just need to represent 2 integers
  - 74123898
  - 4
- But still limited precision due to finite amount of memory available to represent 74123898
  - We speak of the number of significant figures as those that are captured under the representation
  - 74123898→ 4 significant figures → 7412

118

Data representation

Reals: The IEEE 754 Standard

- Consider a number, say, 7412.3898
- Write it as .74123898x10^4
- Now we just need to represent 2 integers
  - 74123898
  - 4
- But still limited precision due to finite amount of memory available to represent 74123898
  - We speak of the number of significant figures as those that are captured under the representation
  - 74123898→ 6 significant figures → 741238

119

Data representation

Reals: The IEEE 754 Standard

- Consider a number, say, 7412.3898
- Write it as .74123898x10^4
- Now we just need to represent 2 integers
  - 74123898
  - 4
- But still limited precision due to finite amount of memory available to represent 74123898
  - We speak of the number of significant figures as those that are captured under the representation
  - 74123898→ 8 significant figures → 74123898

120

Data representation

Java real types

bit	byte	KeyWord	range	precision
32	4	float	+/- 10^38	6 significant figs.
64	8	double	+/- 10^300	15 significant figs.

Data representation

Java real types

bit	byte	KeyWord	range	precision
32	4	float	+/- 10^38	6 significant figs.
64	8	double	+/- 10^300	15 significant figs.

Remark

- In 1020 use double as default
- For example, we might declare (with initialization)

```
double price = 126.37;
```

Data representation

Examples of real representation

- Declaration/initialization:

  float myReal = 54383.27;
- Stored in memory according to our convention as

  543832

  5
- Retrieved from memory as

  54383.2

123

Data representation

Examples of real representation

- Declaration/initialization:

  float myReal = 1000000.0;
- Stored in memory according to our convention as

  1

  7
- Retrieved from memory as

  1000000.0

124

Data representation

Examples of real representation

- Declaration/initialization:

  float myReal = 7412531.0;

- Stored in memory according to our convention as

  741253

  7

- Retrieved from memory as

  7412530.0

125

Data representation

Numbers

- integers
- reals

→ **Characters**

- letters
- digits
- symbols ( ) + -, etc.

Booleans

- true
- false

126

Data representation

Characters

- We choose to represent characters in fashion that is invariant to output matters (e.g., font, etc.)
- So, map characters onto integers and represent them as such.
- Two standard encodings
  - ASCII
  - UNICODE

127

Data representation

ASCII

- Each character allocated 1 byte.
- Provides support for 256 distinct characters.
- Sufficient to capture
  - Digits
  - Various special symbols, e.g., ( ) # $ etc.
  - Letters of English language

128

Data representation

UNICODE

- Each character allocated 2 bytes.
- Provides support for 64000 distinct characters.
- Sufficient to capture
  - Digits
  - Various special symbols, e.g., ( ) # $ etc.
  - Letters of many languages
- First 256 codes are the same as ASCII
- This is the default representation used in Java.
- For details see
  - Roumani textbook, Appendix A

129

Data representation

Examples

- Character declaration

  char grade;
- Character initialization

  grade = 'B';
- "special" characters are dealt with through escape sequences
  - Newline: char startNewLine = '\n';
  - Tab: char insertTab = '\t';
  - Quotation: char singleQuote = '\'';
  - Back slash: char backSlash = '\\';
  - …

130

Data representation

Strings

- A character string is a sequence of 0 or more characters.
- A string can contain a word, a sentence or any amount of text.
- A particular string can be specified as a literal between double quotes.

  "Hello, world!"

Data representation

Strings

- A character string is a sequence of 0 or more characters.
- A string can contain a word, a sentence or any amount of text.
- A particular string can be specified as a literal between double quotes.

  "Hello, world!"

- In Java, character strings are not primitive types (they are object instances of the predefined class String).
  - Generally, objects are used to represent more complex or specialized data than primitive types.
- We will return to this topic latter when we have a bit more machinery in place.

132

Data representation

Numbers

- integers
- reals

Characters

- letters
- digits
- symbols ( ) + -, etc.

→ **Booleans**

- true
- false

133

Data representation

Boolean

- Serve to capture logical true/false values.
- Declaration

```
boolean isBigger;
```

- Initialization

```
isBigger = (a>b);
```

assuming that a and b have been declared and initialized.

Remark

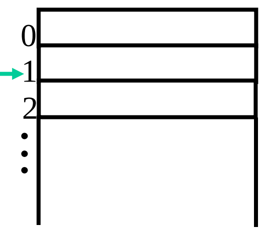
- Note that true and false are Java Keywords.

134

Data representation

Data = Everything we ever want to represent on a computer.

Data representation function

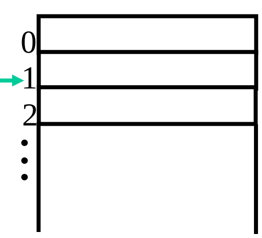


135

Data representation

Data = Everything we ever want to represent on a computer.

Data representation function



Numbers

- integers
- reals

Characters

- letters
- digits
- symbols

Booleans

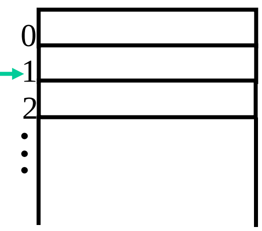
- true
- false

136

Data representation

Data = Everything we ever want to represent on a computer.

Data representation function



Numbers
- integers
- reals

Characters
- letters
- digits
- symbols

Booleans
- true
- false

Remarks
- In Java, the data we manipulate is represented as either a primitive type or an object.
- Almost all that we have seen so far are primitive types.
- The exception is String, a built in class of objects.

137

Variables

What
- Variables are entities in a program that have a value which is allowed to change during the course of the program.

```
int amount, numQuarters;
amount = 78;
numQuarters = amount / 25;
amount = amount – numQuarters * 25;
```

Why
- Variables provide a way to model items with values that vary during the time we interested in them.
- They allow us to abstract away from details of machine representation.

138

Variables

3 Components

- Consider

  int amount = 78;

1. Variables have a type, e.g., int.
   – Can be any Java primitive type (e.g., int, double, boolean, etc.). *Recall we introduced the Java primitive types earlier in these notes.*
   – Can be an object type defined by a class (e.g., String). *We will introduce many more classes as the semester progresses.*

139

Variables

3 Components

- Consider

  int amount = 78;

2. Variables have a symbolic name (identifier), e.g., amount, which is associated with a memory location.
   – Name can be any sequence of letters, digits, $ or _
   – Name cannot begin with a digit.
   – Name cannot be a Java keyword.
   – Choose to be descriptive.

140

Variables

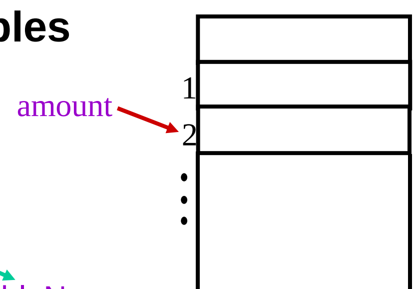
3 Components

- Consider

  int amount = 78;

3. Variables have a value, e.g., 78.
   - This is a value given by you via the program.
   - The value is allowed to change (i.e., vary).
   - The value is what is stored in the memory location associated with the name.

Variables



Declaring a variable

- We abstract

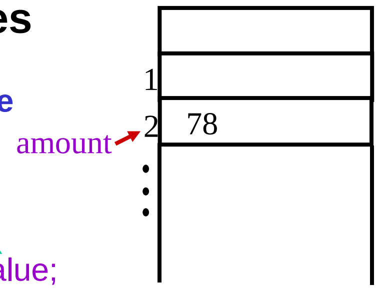
  int amount;

  to

  typeName variableName;

  which is the general Java syntax for variable declaration.
- The declaration
  - Reserves storage space for the variable
  - Associates the (variable) name
  - Specifies what type of values can be stored there.

Variables



Assigning a value to a variable

- We abstract

  amount = 78;

  to

  variableName = value;

  which is the general Java syntax for an assignment statement.
- The assignment statement
  - Places value in the appropriate memory location.
  - LHS must be a variable name.
  - RHS must be a constant, variable or expression.
  - LHS and RHS must be compatible.

143

Variables

Assigning a value to a variable

- We abstract

  amount = 78;

  to

  variableName = value;

  which is the general Java syntax for an assignment statement.
- The assignment statement
  - Places value in the appropriate memory location.
  - LHS must be a variable name.
  - RHS must be a constant, variable or expression.
  - LHS and RHS must be compatible.

144

Variables

Assigning a value to a variable

- We abstract

  amount = yestedayAmount;

  to

  variableName = value;

  which is the general Java syntax for an assignment statement.
- The assignment statement
  - Places value in the appropriate memory location.
  - LHS must be a variable name.
  - RHS must be a constant, variable or expression.
  - LHS and RHS must be compatible.

145

Variables

Assigning a value to a variable

- We abstract

  amount = 78 + yesterdayAmount;

  to

  variableName = value;

  which is the general Java syntax for an assignment statement.
- The assignment statement
  - Places value in the appropriate memory location.
  - LHS must be a variable name.
  - RHS must be a constant, variable or expression.
  - LHS and RHS must be compatible.

146

Variables

Assigning a value to a variable

- We abstract

  amount = 78 + yesterdayAmount;

  to

  variableName = value;

  which is the general Java syntax for an assignment statement.
- The assignment statement
  - Places value in the appropriate memory location.
  - LHS must be a variable name.
  - RHS must be a constant, variable or expression.
  - LHS and RHS must be compatible.

Variables

Assigning a value to a variable

- We abstract

  amount = 78 + yesterdayAmount;

  to

  variableName = value;

  which is the general Java syntax for an assignment statement.

Remark

- Here = does not denote equality.

Variables

Initialization

- A variable must be given a value before we use it.
- Giving a variable its first value is called initialization.
- It is possible to combine declaration with initialization.
- We abstract

int amount = 78;

to

typeName variableName = value;

149

Variables

Why use variables (again)

- Variables allow us to model malleable data.
- They allow us to abstract away from machine representation.
- They are keys to
  - Understandable software.
  - Maintainable software.

150

Constants

What

- Constants are entities in a program that have value that does not change.

  final int DAYS_IN_A_YEAR = 365;

  final double EARTH_ESCAPE_VELOCITY = 11.2; // km/sec
- Similar to variable declaration/initialization with addition of keyword final.

Why

- Allows us to *avoid* magic numbers in our programs.
  - Numbers that appear without explanation.
- Improves program
  - readability
  - maintainability.

151

Arithmetic expressions

Basic operators work on the number types much as expected

```
final double SCALE_F2C = 5.0 / 9.0;
final double OFFSET_F2C = 32.0;
double degF = 100.0;
double degC = SCALE_F2C * (degF – OFFSET_F2C);
```

Remarks

- Use * for multiplication
- Use % for remainder

  remains = 13 % 5; // remains equals 3

152

Arithmetic expressions

Basic operators work on the number types much as expected

```
final double SCALE_F2C = 5.0 / 9.0,
final int OFFSET_F2C = 32.0;
double degF = 100.0;
double degC = SCALE_F2C * (degF – OFFSET_F2C);
```

Remarks

- The division operator performs integer division (discarding the remainder) when both its arguments are integer types; otherwise it does real division
  - 13 / 5 evaluates to 2
  - 13.0 / 5.0, 13.0 / 5, 13 / 5.0 all evaluate to 2.6

153

Arithmetic expressions

Basic operators work on the number types much as expected

```
final double SCALE_F2C = 5.0 / 9.0,
final int OFFSET_F2C = 32.0;
double degF = 100.0;
double degC = SCALE_F2C * (degF – OFFSET_F2C);
```

Remarks

- Arithmetic operations with floating point number types are rarely exact
  - (1.0 / 3.0) + (2.0 / 3.0) will evaluate to a value (slightly) less than 1.0
  - Recall discussion on floating point representation

154

Arithmetic expressions

Precedence

operator	precedence
unary -	high
* / %	medium
+ binary -	low

- Within a precedence group evaluation is left to right.
- Parentheses ( ) force what they enclose to be evaluated first.

```
int example = 14 – 8 / 2 + 1; // 11
int example = (14 – 8) / (2 + 1) ; // 2
```

- See textbook Appendix B for extended precedence table.

155

Arithmetic expressions

Other operators

- Java also provides

```
int demo = 2; // demo equals 2
demo++; // demo equals 3
demo--;  // demo equals 2
demo += 23; // demo equals 25
```

etc.

- Further, in the Math class we find lots of goodies, e.g., Math.sqrt(x), Math.sin(x), …

156

Arithmetic expressions

More generally

- Here we have concentrated on arithmetic expressions.
- However, we think of an expression as anything that can be evaluated to yield a value.

157

Type promotion and casting

Promotion

- A value can only be assigned to a variable if they are of the same type.
- An operator or method can only be applied to the type of data on which it is defined.
- Java will automatically promote a value from a smaller to a larger numeric type

```
double eg = 2.5 + 3; // 3 promoted to double, eg value is 5.5
```

- Similarly, if you call a method that takes a double argument type with an int argument, then the argument will be promoted to a double.

```
int x = 3;
double y = Math.sin(x); // x promoted to double
```

158

Type promotion and casting

Automatic promotion rules

- To int in any non-long integer mix
- To long if there's a long in an integer mix
- To float if there's a float in a non-double mix
- To double if there's a double in any mix

159

Type promotion and casting

Casting

- A value of a larger type is never automatically converted to a value of a smaller type.
  - Risk of information loss.
- To avoid a mismatch error, we must use a type cast.

```
double x = 3.5;
int n = (int) x; // cast the double x as an int
```

- The value 3.5 is truncated to 3; the fractional part is discarded.
- The general (Java) syntax for a type cast is

```
variableOfTypeName = (typeName) expression;
```

160

Type promotion and casting

Rounding

- Often it is desirable to round up prior to casting to an integer type

```
double x = 3.5;
int n = (int) (x + 0.5); // n = 4
int n = (int) Math.round(x); // n =4
```

161

Type promotion and casting

Precedence

- A cast works on the variable immediately to its right,
- i.e., similar to the prefix unary operator –.

162

Type promotion and casting

Remarks

- Java is strict about type agreement.
- It supports promotion and explicit casting, the second for when there is risk of loosing information.
- Some other languages are strict about type agreement, but do not support promotion and casting (bondage and discipline languages).
- Some languages do not type check (shoot yourself in the foot languages).

163

Type promotion and casting

Remarks

- Java is strict about type agreement.
- It supports promotion and explicit casting, the second for when there is risk of loosing information.
- Some other languages are strict about type agreement, but do not support promotion and casting (bondage and discipline languages).
- Some languages do not type check (shoot yourself in the foot languages).
- 1020 students should make up many computer-based examples for themselves to ensure understanding of promotion and casting in Java.

164

Type promotion and casting

Review by way of examples

```
byte myByte, yourByte;
myShort, yourShort;
int yourInt;
yourByte = myByte; // error
myByte = 200; // error
myByte = 100; // okay
myShort = 200; // okay
yourByte = myByte + 1; // error
myShort = myByte + 1; // error
yourInt = myByte +1; // okay
yourByte = (byte) myByte + 1; // error
yourByte = (byte) (myByte + 1); // okay
```

165

Type promotion and casting

Review by way of examples

```
byte myByte, yourByte;
myShort, yourShort;
int yourInt;
yourByte = myByte; // error
myByte = 200; // error
myByte = 100; // okay
myShort = 200; // okay
yourShort = myByte + myShort; // error
yourByte = 50; // okay
yourShort = myByte + yourByte; // error
yourShort = (short) (myByte + yourByte); // okay
```

166

Type promotion and casting

Review by way of examples

```
int score1 = 5; score2 = 6; score3 = 3;
double average = (score1 + score2 + score3) / 3;
println(average); // prints 4.0
average = (score1 + score2 + score3) / 3.0;
println(average); // prints 4.66 … 7
```

167

Type promotion and casting

Review by way of examples

```
double total;
int dollars = 2; // okay
total = “a lot”; // error
total = dollars; // okay
dollars = total; // error
dollars = (int) total; // okay
total = 13.75; // okay
int pennies = (int) (total * 100); // 1375
int pennies = (int) total * 100; // 1300
```

168

Type promotion and casting

Review by way of examples

```
float myFloat, yourFloat;
myFloat = 0.1; // error
myFloat = (float) 0.1; // okay
yourFloat = myFloat; // okay
yourFloat = myFloat + 0.1; // error
yourFloat = (float) myFloat + 0.1; // error
yourFloat = (float) (myFloat + 0.1); // okay
```

169

Type promotion and casting

Review by way of examples

•It is critical that 1020 students generate additional examples for self evaluation.

1. Make “theoretical predictions” based on applicable rules.
2. Validate via computational “experiments”.

•Repeat steps 1 and 2 until all examples are understood.

170

Summary

- Abstraction
- Hardware abstraction
- Software abstraction
- Data abstraction

171