

3.2 Impact of Program Design

Self-Study Guide · About 45 minutes · With answer key

By the end you will be able to

- Describe ways a program can affect society, the economy, and culture, and explain why the same program can help some people and harm others.
- Explain how a program built to solve one problem can cause harm its designers never intended.
- Define system reliability and explain what "stated conditions" means for a program.
- Choose a variety of test cases, including boundary and special cases, to find failures before users do.
- Explain what intellectual property is and why using someone else's code can raise legal issues.
- Tell when code can be reused freely (open source) and when a programmer must get permission and often pay.

What to remember from earlier

- A logic error runs to the end with the wrong result. ([Topic 1.1: Introduction to Algorithms, Programming, and Compilers, 1.1.7 Logic Errors and Testing](#))
- Testing means running a program with specific data and comparing the result to the expected outcome. ([Topic 1.1: Introduction to Algorithms, Programming, and Compilers, 1.1.7 Logic Errors and Testing](#))
- A run-time error, such as dividing an int by zero, stops a program early. ([Topic 1.1: Introduction to Algorithms, Programming, and Compilers, 1.1.8 Run-Time Errors and Exceptions](#))
- Procedural abstraction allows code reuse. ([Topic 3.1: Abstraction and Program Design, 3.1.6 Generalizing with Parameters](#))

3.2.1 Programs Affect People

So far, every program in this course has lived on your own screen. If it printed the wrong thing, you were the only one who saw it. Real programs are different. Once a program is released, thousands of people may depend on it to get to school, buy food, do their jobs, or talk to friends. Whatever the program does, good or bad, it does to all of them.

Programmers therefore have to think about more than "does it compile?" They also have to ask "what will this do to the people who use it, and to the people around them?" An **impact** of a program is any effect it has on people beyond the screen. It helps to sort impacts into three areas:

Area	What it covers	Example question to ask
Society	How people live, learn, stay safe, and treat others	Who can use it? Who is left out?
Economy	Money, jobs, businesses, and costs	Who earns or saves money? Who loses work?

Culture	Shared habits, language, art, and traditions	How does it change what people do or care about?
----------------	--	--

The key point is that impacts come in two kinds. A **beneficial** impact makes things better for someone; a **harmful** impact makes things worse. One program usually has both, often for different groups of people. That is why "is this program good?" rarely has a one-word answer.

KEY IDEA

Creating a program affects society, the economy, and culture. Those effects can be beneficial, harmful, or both at once, and they often land on different groups of people.

WORKED EXAMPLE Sorting the impacts of a lunch pre-order app

Imagine a school releases a hypothetical app called LunchLine. Students choose and pay for lunch on their phones before noon, then pick it up at a fast lane. Go through the three areas one at a time and ask who is helped and who is hurt.

Area	Beneficial impact	Harmful impact
Society	Shorter lines, so students get more time to eat	Students without a phone or a payment card wait in a slower line
Economy	The cafeteria wastes less food because it knows orders	The cafeteria needs fewer cashier hours, so some workers lose shifts
Culture	Menus can show foods from many cultures with pictures	Lunch becomes less social when students eat apart from friends in line

Notice how the analysis worked: for each area, name a specific group and a specific change for that group. "It is convenient" is too vague to be useful. "Students without phones wait longer" names who is affected and how.

Now change one thing: the group you look at. For a student with a phone, LunchLine is almost all benefit. For a cashier, it may be mostly harm. For a student without a phone, it can turn lunch from equal for everyone into slower for some. The program did not change; the person did. Impact always depends on whose point of view you take, so a careful programmer asks the question from several points of view.

WATCH OUT

"My program helps people, so it has no harmful impacts." Students often stop after listing the benefits, because the benefits are why they wrote the program. But a benefit for one group does not cancel a harm for another. Always ask the second question: who might be worse off because this exists?

PAUSE AND TRY**Question 1** Multiple choice

A town releases a hypothetical app that lets residents report potholes with a photo. Repair crews now fix roads faster, and the town spends less money on inspectors driving around looking for damage. Which statement best describes these impacts?

- A. Both are social impacts, because the app is used by residents.
- B. Faster repairs are a social benefit; spending less on inspectors is an economic impact.
- C. Both are cultural impacts, because the app changes a town habit.
- D. Neither is an impact, because the app does not cause any harm.

Question 2 Short answer

A hypothetical homework app reads math problems aloud for students who have trouble reading small print. Name one beneficial impact and one possible harmful impact. For each, say who is affected.

3.2.2 Unintended Harmful Effects

Some harms are easy to predict before a program is released. Others surprise everyone, including the people who wrote the program, because they come from people using it in ways its designers never pictured.

Every program is written to fill a need or solve a problem. That goal is its **intended use**. But once a program is in the world, people use it however they like, and the program does exactly what its code says no matter who is using it or why. An **unintended effect** is a result the designers did not plan or expect. When it hurts someone, it is an unintended harmful effect, and it can happen even when everyone who built the program had good intentions.

Think of a kitchen knife. It is designed to cut food, and almost everyone uses it that way. The designer never meant for it to be used to hurt anyone, yet a knife can still cause harm. Programs are similar, with one difference that makes them riskier: a program can reach millions of people at once, so a small unintended effect can grow very large very fast.

KEY IDEA

A program made to fill a need or solve a problem can cause harm beyond its intended use. Good intentions do not prevent unintended harmful effects, so designers should look for them before and after release.

WORKED EXAMPLE A shortcut that moved the traffic

A hypothetical navigation app is written to solve one problem: get each driver to their destination as quickly as possible. Trace how its intended use turns into an unintended harm.

Step	What happens	Intended?
------	--------------	-----------

1	The main road is jammed, so the app finds a faster route through a quiet neighborhood	Yes: faster trips
2	The app gives the same suggestion to every driver on the jammed road	Yes: each driver is helped
3	Hundreds of cars now pour down a narrow street lined with houses	No
4	Children playing near the street are less safe, and residents cannot leave their driveways	No: a harm to people who never used the app

Each step follows from the one before, and no line of code was "wrong." The program did exactly what it was designed to do for each driver. The harm came from what happens when the program does that for everyone at once, and it fell on people who never opened the app.

Now vary one thing: the person using the program. A hypothetical app that shows the live location of a delivery driver is intended to tell customers when food will arrive. Used by a customer, it does just that. But the same feature, in the hands of someone who wants to follow the driver, could put the driver at risk. Here the unintended harm comes not from many users but from one user with a different goal. Asking "how could someone misuse this?" is part of designing a program responsibly.

WATCH OUT

"If I did not mean for it to happen, it is not my program's impact." Students often think only intended effects count. But the people hurt by the shortcut traffic are hurt whether or not the designers meant it. Programmers cannot predict everything, but they are expected to ask how their program could be used beyond its purpose and to fix harms once they learn about them.

PAUSE AND TRY

Question 3 Multiple choice

A hypothetical study app awards points for every minute a student keeps it open, to encourage studying. Which outcome is the best example of an unintended harmful effect?

- A. Students spend more time studying.
- B. Some students leave the app open all night to earn points, and they lose sleep.
- C. The app does not compile because of a missing semicolon.
- D. Students who like the app tell their friends about it.

Question 4 Short answer

A hypothetical app lets neighbors post photos of "suspicious" people in their area so they can keep each other safe. Describe one unintended harmful effect, and explain why it is unintended.

3.2.3 System Reliability

Some harm happens simply because a program does not work. If a bus-tracking app says the bus is ten minutes away when it is actually at the stop, a student misses school. People can only depend on a program that does its job every time they need it.

Recall from Topic 1.1 that a logic error gives a wrong result and a run-time error stops a program early. One error found by one tester is a small problem. The same error in a program that thousands of people depend on is a big one.

System reliability means a program can do the tasks it is supposed to do, as expected, under its **stated conditions**, without failing. The stated conditions are the situations the program is promised to handle: the kinds of input, the number of users, the devices it runs on, and so on. A reliable program does not have to do everything. It has to do what it promised, in every situation it promised to cover, every time.

KEY IDEA

A reliable program performs its tasks as expected under its stated conditions, without failure. Working once, or working for typical input, is not the same as being reliable.

WORKED EXAMPLE A season report that crashes

A hypothetical team app shows each player's points per game. Its description promises a report "for every player on the roster." A tester first tries a typical player who scored 45 points in 3 games:

```
1 int totalPoints = 45;
2 int gamesPlayed = 3;
3 System.out.println("Season report");
4 int pointsPerGame = totalPoints / gamesPlayed;
5 System.out.println("Points per game: " + pointsPerGame);
```

```
Season report
Points per game: 15
```

The integer division $45 / 3$ gives 15, and the program works. Now the tester tries a new player who just joined the roster and has not played yet, so both values are 0:

```
1 int totalPoints = 0;
2 int gamesPlayed = 0;
3 System.out.println("Season report");
4 int pointsPerGame = totalPoints / gamesPlayed;
5 System.out.println("Points per game: " + pointsPerGame);
```

```
Season report
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

The first line prints, then $0 / 0$ is an integer division by zero, which throws an `ArithmeticException` and stops the program. (The exact wording of the message can vary between Java versions.) A new player is on the roster, so this is inside the stated conditions. The program failed at something it promised to do, which means it is not reliable yet.

The fix uses an `if` statement from Unit 2 so the division happens only when it is safe:

```
1 int totalPoints = 0;
2 int gamesPlayed = 0;
3 System.out.println("Season report");
4 if (gamesPlayed > 0)
5 {
6     int pointsPerGame = totalPoints / gamesPlayed;
7     System.out.println("Points per game: " + pointsPerGame);
8 }
9 else
10 {
11     System.out.println("No games played yet");
12 }
```

```
Season report
No games played yet
```

Stated conditions work in both directions. If this app's description had said "for players who have played at least one game," a player with zero games would be outside the stated conditions. Even then, a careful programmer would handle it, because real users rarely read the fine print. Clear stated conditions tell everyone what the program must do; reliability is meeting that promise.

WATCH OUT

"It worked when I ran it, so it is reliable." Running a program once, with the first values that come to mind, usually tests only the typical case. The season report worked perfectly for 45 points in 3 games and crashed for a brand-new player. Reliability is about every situation the program promises to handle, not the one you happened to try.

PAUSE AND TRY

Question 5 Multiple choice

A hypothetical alarm clock app states that it works on phones with the latest system update. Which situation shows the app is **not** reliable?

- A. It does not run on a ten-year-old phone that cannot install the latest update.
- B. On an up-to-date phone, it sometimes fails to ring when the alarm time arrives.
- C. It does not include a weather forecast.
- D. A user sets the alarm for the wrong time, and it rings at that wrong time.

Question 6 Tracing

A teammate tests the **fixed** season report with a player who scored 7 points in 2 games. What does it print? Which branch of the `if` runs?

```

1  int totalPoints = 7;
2  int gamesPlayed = 2;
3  System.out.println("Season report");
4  if (gamesPlayed > 0)
5  {
6      int pointsPerGame = totalPoints / gamesPlayed;
7      System.out.println("Points per game: " + pointsPerGame);
8  }
9  else
10 {
11     System.out.println("No games played yet");
12 }

```

3.2.4 Testing With a Variety of Conditions

Sooner or later, a program's users will run into situations its programmer never tried. Programmers want to find the failures first, and their main tool is testing on purpose with many different conditions before the program is released.

Recall that **testing** means running a program with specific data and comparing the actual result to the expected outcome. The question now is which data to choose. Picking several values that are all alike ("8, 9, 10") tells you almost nothing new after the first one. Good test data is chosen to cover different situations:

Kind of test case	What it checks	Example (from this chapter's programs)
Typical case	Ordinary values in the middle of each group	8, 30, 70
Boundary case	Values exactly at, or right next to, where a rule changes	12 and 13; 64 and 65
Special case	Values that might need different handling, such as zero	0 games played

Boundary cases matter because, as you saw in Unit 2, `<` versus `<=` changes the result only at the boundary. A program with that mistake passes every typical test and fails only when the value lands exactly on the line.

KEY IDEA

Programmers make programs more reliable by testing them with a variety of conditions: typical values, boundary values, and special cases. Each kind of test can catch errors the others miss.

WORKED EXAMPLE Testing a ticket price program

A hypothetical theater charges \$6 for children 12 and under, \$10 for ages 13 to 64, and \$7 for ages 65 and over. The program below reads four test ages from a `Scanner` built from a string (it works the same way as typed input) and prints the price for each.

TicketPriceTests.java

```
1 import java.util.Scanner;
2
3 public class TicketPriceTests
4 {
5     public static void main(String[] args)
6     {
7         Scanner testAges = new Scanner("8 30 70 12");
8         for (int test = 1; test <= 4; test++)
9         {
10             int age = testAges.nextInt();
11             int price;
12             if (age < 12)
13             {
14                 price = 6;
15             }
16             else if (age < 65)
17             {
18                 price = 10;
19             }
20             else
21             {
22                 price = 7;
23             }
24             System.out.println("Age " + age + ": $" + price);
25         }
26     }
27 }
```

```
Age 8: $6
Age 30: $10
Age 70: $7
Age 12: $10
```

Compare each result to the expected outcome:

Test age	Kind of case	Expected	Actual	Pass?
8	Typical	\$6	\$6	Yes
30	Typical	\$10	\$10	Yes
70	Typical	\$7	\$7	Yes
12	Boundary	\$6	\$10	No

All three typical tests pass. Only the boundary test fails: line 12 checks `age < 12`, which is `false` when `age` is 12, so a 12-year-old falls through to the adult price. That is a logic error that a tester using only typical values would never see.

Now vary the condition on line 12 to `age <= 12`, and add more boundary tests on both sides of both rule changes:

TicketPriceFixed.java

```
1 import java.util.Scanner;
2
3 public class TicketPriceFixed
4 {
5     public static void main(String[] args)
6     {
7         Scanner testAges = new Scanner("8 30 70 12 13 64 65");
8         for (int test = 1; test <= 7; test++)
9         {
10             int age = testAges.nextInt();
11             int price;
12             if (age <= 12)
13             {
14                 price = 6;
15             }
16             else if (age < 65)
17             {
18                 price = 10;
19             }
20             else
21             {
22                 price = 7;
23             }
24             System.out.println("Age " + age + ": $" + price);
25         }
26     }
27 }
```

```
Age 8: $6
Age 30: $10
Age 70: $7
Age 12: $6
Age 13: $10
Age 64: $10
Age 65: $7
```

Every result now matches the price table. The tests at 64 and 65 show that the second rule, `age < 65`, was already correct. Testing both sides of each boundary is how you know a rule changes in exactly the right place, not one value early or late.

WATCH OUT

Testing many values that are all alike. Students often test 5, 6, 7, 8, and 9 and feel thorough. Those five tests are really one test repeated: all typical children. One typical value per group plus the values on each side of every boundary, plus any special case such as zero, tests far more with fewer runs.

PAUSE AND TRY**Question 7** Multiple choice

A hypothetical library program says: "Books more than 14 days late cost \$1; books 14 days late or less cost nothing." Which set of test values best checks the program?

- A. 1, 2, 3, 4
- B. 20, 25, 30, 35
- C. 3, 14, 15, 30
- D. 14, 14, 14, 14

Question 8 Debugging

A hypothetical store gives free shipping on orders of **\$50 or more**. The tester runs three orders:

ShippingTests.java

```
1 import java.util.Scanner;
2
3 public class ShippingTests
4 {
5     public static void main(String[] args)
6     {
7         Scanner testOrders = new Scanner("20 80 50");
8         for (int test = 1; test <= 3; test++)
9         {
10             int orderTotal = testOrders.nextInt();
11             if (orderTotal > 50)
12             {
13                 System.out.println(orderTotal + ": free shipping");
14             }
15             else
16             {
17                 System.out.println(orderTotal + ": shipping costs $5");
18             }
19         }
20     }
21 }
```

```
20: shipping costs $5
80: free shipping
50: shipping costs $5
```

Which test case reveals the error? What kind of test case is it? Correct the line that has the error.

3.2.5 Intellectual Property and the Law

Creating a program raises questions not only about what it does to its users, but about the code itself: who owns it, and who is allowed to copy it?

Writing a program takes real work and creativity, just like writing a song or a novel. The law treats it the same way. **Intellectual property** is something a person creates with their mind, such as a song, a drawing, a story, an invention, or a program. The creator, or the company they made it for, has legal rights over it, including the right to decide who may copy it, use it, or sell it.

That means programmers face **legal issues** whenever they use code they did not write. Code you can see on a website is not automatically free to take, any more than a song you can hear online is free to put in your own album. Copying someone else's code into your program without permission can break the law, even if you change a few names.

KEY IDEA

Code is intellectual property. Its creator has rights over how it is copied and used, so using someone else's code raises legal questions that must be answered before the code goes into your program.

WORKED EXAMPLE Who owns the code?

Look at each situation and ask: who created the code, and did the programmer have permission to use it?

Situation (all hypothetical)	Legal concern?
Maya writes a quiz program from scratch for her own project	No: Maya created it, so it is her intellectual property
Maya copies a game's scoring code from a company's website, where it is shown but not offered for use	Yes: the company owns it, and Maya has no permission
Maya uses <code>Math.sqrt</code> and <code>String</code> methods from Java's own library	No: Java's library is provided so that Java programmers can use it in their programs

The deciding question in every row is the same: not "could I get the code?" but "was I given permission to use it?"

Now vary one detail of the second row: suppose the company's website says "you may use this code in your own programs." The code is identical, and so is the owner, but now Maya has permission, so the legal concern is gone. What makes code safe to reuse is not where you found it but what its owner allows.

WATCH OUT

"If it is online, it is free to use." Students often assume anything they can find with a search is fair game. Being easy to find is not the same as being offered for use. Before you copy code, look for a statement from its owner that says what you are allowed to do with it.

PAUSE AND TRY

Question 9 Multiple choice

Which statement about intellectual property is accurate?

- A. Only physical objects, such as machines, can be intellectual property.
- B. Code becomes free for anyone to use once it is posted on a website.
- C. A program is intellectual property, and its owner decides who may copy and use it.
- D. Changing the variable names in copied code makes it your own intellectual property.

3.2.6 Reusing Code: Open Source and Permission

In Topic 3.1 you saw that reusing code, instead of writing the same thing again, helps manage complexity. Professional programmers reuse code written by other people all the time, yet that code is someone else's intellectual property. So how do they reuse it legally?

There are two main routes, and which one applies depends on how the code's owner published it.

Open source code is code whose owner has published it so that anyone may use it for free. Programmers can put open-source code into their own programs without asking first. Open-source code comes with a **license**, a short statement of what users may do and any simple conditions to follow, such as keeping the original author's name in the code. Following the license is part of using the code legally.

Code that is **not** published as open source belongs to its owner in the ordinary way. To put it into your program, you must **get permission** from the owner, and often you must **purchase** the right to use it before you include it.

KEY IDEA

Open-source code is published free to use, so programmers may reuse it by following its license. Code that is not open source requires the owner's permission, and often a purchase, before it goes into a program.

WORKED EXAMPLE Two routes to the same feature

A hypothetical student team is building a fitness app and needs code that draws charts. They find two options.

Option	Open source?	What the team must do
A chart library published as open source, with a license asking users to keep the authors' names	Yes	Use it for free and follow the license by keeping the authors' names
A chart package sold by a hypothetical software company	No	Contact the company, get permission, and likely pay before using it

Either option can be used legally. The difference is the steps the team takes first: for the open-source library, read and follow the license; for the company's package, get permission and probably pay.

Now vary the team's choice: suppose they skip both steps and copy a chart function out of the company's package because a classmate shared the file. The code works just as well, but the team used intellectual property without permission. If the app is released, the company could take

legal action. Getting the feature is never the whole question; getting it the right way is part of the job.

WATCH OUT

"Open source means no rules at all." Open-source code is free to use, but it still has an owner and a license. Students sometimes delete the author's name from open-source code or ignore its conditions. Read the license and follow what it asks.

PAUSE AND TRY**Question 10** Multiple choice

A programmer wants to use code that a hypothetical company sells and has **not** published as open source. What must the programmer do before putting it in their program?

- A. Nothing, as long as the program is free for users.
- B. Change enough of the code that it no longer looks like the original.
- C. Get permission from the company, and likely purchase the right to use it.
- D. Mention the company's name in the program's output.

Question 11 Short answer

A classmate says, "Open-source code and code a company sells are both written by other people, so the rules for using them must be the same." Explain why the classmate is wrong.

Putting it together

A hypothetical student, Dev, builds a parking app for a busy school. Students check how many parking spots are open in each lot before they leave home. Follow Dev through every idea in this chapter.

Reusing code (3.2.5, 3.2.6). Dev needs a map to show the lots. He finds an open-source map library whose license asks users to keep the authors' names, and a fancier map package sold by a hypothetical company. He picks the open-source library and keeps the authors' names in his code, as its license asks. Had he chosen the company's package, he would have needed the company's permission and probably a purchase first.

Reliability and testing (3.2.3, 3.2.4). Dev's description states that the app shows the open spots in each lot, which hold up to 50 cars. He writes down his stated conditions, then chooses test cases on purpose:

Test	Kind of case	Why Dev chose it
20 cars in the lot	Typical	An ordinary morning
49 and 50 cars	Boundary	The app must switch to "Lot full" at exactly 50
0 cars (early morning)	Special	An empty lot is an unusual value that might need different handling

His first version still shows an open spot when exactly 50 cars are parked, because one comparison used `<=` where it needed `<`. Only the boundary test exposes this logic error. He fixes it before anyone relies on the app to decide where to park.

Impacts (3.2.1). Socially, students spend less time circling the lots, and fewer late arrivals mean fewer missed first periods. Economically, the school may need fewer staff directing traffic. Culturally, the morning routine changes: students leave home based on an app instead of habit.

Unintended harm (3.2.2). After launch, students who see "Lot A: 2 spots" all race to Lot A at once, making the driveway more dangerous than before. Dev never intended that. He responds by showing ranges ("a few spots") instead of exact counts, and by testing what happens when many users get the same information at once.

Every part of this story involves a choice about how the program is designed: which code to reuse, which tests to run, and how the information is shown. Those design choices are what decide the program's impact.

Vocabulary

Term	Definition	Example
impact	Any effect a program has on people beyond the screen, in society, the economy, or culture	Shorter lunch lines
society / economy / culture	How people live and treat each other / money, jobs, and businesses / shared habits, language, and art	Safety / lost cashier shifts / changed lunch routines
beneficial / harmful	An impact that makes things better / worse for someone	Faster trips / traffic on a quiet street
intended use	The need or problem a program was written to address	Getting drivers somewhere faster
unintended effect	A result its designers did not plan or expect; harmful ones go beyond the intended use	Shortcut traffic through a neighborhood
system reliability	A program doing its tasks as expected, under its stated conditions, without failure	An alarm that always rings on time
stated conditions	The situations a program is promised to handle	"For every player on the roster"
testing	Running a program with chosen data and comparing actual results to expected outcomes	Expect \$6 for age 12
typical / boundary / special case	An ordinary value / a value at or next to where a rule changes / a value needing different handling	30 / 12 and 13 / 0 games played

intellectual property	A creation of the mind, such as a program, whose creator has rights over its copying and use	A company's scoring code
legal issues	Questions of law that arise when creating programs, such as using code you did not write	Copying code without permission
open source	Code published so anyone may use it for free	A free chart library
license	The owner's statement of what users may do with code and any conditions to follow	"Keep the authors' names"

Chapter summary

- Creating a program has **impacts** on society, the economy, and culture, and the same program can be **beneficial** for some people and **harmful** for others.
- A program written to fill a need can cause **unintended harmful effects** beyond its intended use, even when its designers meant well.
- **System reliability** means a program performs its tasks as expected under its **stated conditions** without failure.
- Programmers maximize reliability by **testing** with a variety of conditions, especially **boundary** and **special cases** that typical values miss.
- Code is **intellectual property**. **Open-source** code may be reused for free by following its **license**; other code requires the owner's permission and often a purchase.

Review questions

Question 12 Multiple choice

A hypothetical video app automatically adds captions to every video. It works well for clear speech in English, which its description promises, but it often produces nonsense captions for people with strong accents. Which statement is most accurate?

- A. The app is unreliable for clear English speech, so it should not be released.
- B. The app has a beneficial social impact for viewers who cannot hear, but it can harm speakers with accents whose words are shown wrongly.
- C. Because captions help people, the app has no harmful impacts.
- D. The nonsense captions are a syntax error that the compiler should have caught.

Question 13 Tracing

A hypothetical thermostat program should turn the heat on when the temperature is **68 degrees or lower**. A tester runs it for temperatures 66 through 69. What does it print, and does it meet its specification?

```
1 for (int temperature = 66; temperature <= 69; temperature++)
2 {
3     if (temperature < 68)
4     {
5         System.out.println(temperature + ": heat on");
6     }
7     else
8     {
9         System.out.println(temperature + ": heat off");
10    }
11 }
```

Question 14 Short answer

A hypothetical app helps farmers decide when to water crops. In testing, the programmers used only data from rainy spring weeks, and every test passed. Explain why the app might still not be reliable, and name one test condition they should add.

Question 15 Multiple choice

A hypothetical app that tells people the cheapest gas station nearby becomes very popular. Within weeks, the cheapest station has long lines that block a busy road. Which phrase best describes the blocked road?

- A. A reliability failure, because the app gave wrong prices
- B. An unintended harmful effect beyond the app's intended use
- C. A legal issue, because the station's prices are intellectual property
- D. A beneficial economic impact for all drivers

Question 16 Short answer

A student building a free game for a class project finds three pieces of code: (1) an open-source sound library whose license asks users to keep its copyright notice, (2) a physics engine a company sells, and (3) a menu function on a hobbyist's website with no statement about who may use it. For each, say whether the student may use it right away and what, if anything, the student must do.

Reflection

Check each statement. If the answer is "not yet," reread the linked subsection.

- ☐ I can describe beneficial and harmful impacts of a program on society, the economy, and culture. Reread: 3.2.1 Programs Affect People
- ☐ I can explain how a program can cause harm beyond its intended use. Reread: 3.2.2 Unintended Harmful Effects
- ☐ I can define system reliability and explain what stated conditions are. Reread: 3.2.3 System Reliability
- ☐ I can choose typical, boundary, and special test cases for a program. Reread: 3.2.4 Testing With a Variety of Conditions
- ☐ I can explain why using someone else's code raises legal and intellectual property issues. Reread: 3.2.5 Intellectual Property and the Law

☐ I can tell when code may be reused freely and when permission or purchase is needed. Reread:
3.2.6 Reusing Code: Open Source and Permission

Check Your Reasoning

Answers and explanations for every question, numbered to match.

Question 1

B. Safer, better roads change how people live day to day, which is a social effect. Spending less money on inspectors is about money and jobs, which is an economic effect. (For the inspectors whose work is cut, that same change may be harmful.)

- **A** is wrong: who uses an app does not decide the area. Money saved is an economic effect.
- **C** is wrong: neither effect is mainly about shared habits, language, or traditions.
- **D** is wrong: an impact is any effect on people, helpful or harmful. Benefits count as impacts too.

If you chose differently, reread *3.2.1 Programs Affect People*.

Question 2

Model answer: Beneficial: students with vision difficulties can work on the same assignments as their classmates without waiting for help (society). Harmful: students whose families cannot afford a device that runs the app are left out, so the gap between them and other students grows (society, and economy because of the device cost).

A complete answer must:

- name one benefit and one harm, and
- name a specific group of people for each.

Other good harms include students relying on the app instead of building reading skills, or a company selling paper study guides losing customers (economy). If you named only benefits, reread *3.2.1 Programs Affect People*.

Question 3

B. The designers wanted points to reward studying. Students using the points in a way the designers did not picture, leaving the app open without studying, is beyond the intended use, and lost sleep is a harm.

- **A** is wrong: more studying is the intended, beneficial effect.
- **C** is wrong: a syntax error is a problem with the code, not an effect of a released program on people. A program that does not compile never reaches users.
- **D** is wrong: this is not a harm.

If you chose differently, reread *3.2.2 Unintended Harmful Effects*.

Question 4

Model answer: People might post photos of innocent neighbors or visitors based on how they look, and those people could be harassed or treated unfairly. It is unintended because the app was built to improve safety, not to single out innocent people, but the app posts whatever users choose to post.

A complete answer must:

- describe a specific harm to specific people, and
- explain that it goes against or beyond the app's purpose (keeping neighbors safe).

If you described a benefit or a bug in the code, reread *3.2.2 Unintended Harmful Effects*.

Question 5

B. Ringing at the set time is the app's main task, and an up-to-date phone is inside its stated conditions. Failing there is a reliability failure.

- **A** is wrong: an old phone that cannot update is outside the stated conditions, so not running there does not break the app's promise.
- **C** is wrong: a weather forecast is not one of the app's tasks. Reliability is about doing promised tasks, not every possible task.
- **D** is wrong: the app did exactly what it was told. The mistake was the user's, not the program's.

If you chose differently, reread *3.2.3 System Reliability*.

Question 6

The output is:

```
Season report
Points per game: 3
```

Statement	What happens
<code>System.out.println("Season report");</code>	Prints Season report
<code>if (gamesPlayed > 0)</code>	<code>2 > 0</code> is true, so the first branch runs
<code>totalPoints / gamesPlayed</code>	<code>7 / 2</code> is integer division: 3, not 3.5
<code>System.out.println(...)</code>	Prints Points per game: 3
<code>else branch</code>	Skipped

The guard does not change typical results; it only keeps the division from running when `gamesPlayed` is 0. If you wrote 3.5, remember that dividing two `int` values drops the fractional part. Reread *3.2.3 System Reliability* if you expected the `else` branch.

Question 7

C. It includes a typical value from each group (3 and 30) and the values on both sides of the boundary (14 and 15), where a `>` versus `>=` mistake would show up.

- **A** is wrong: all four are typical "no fine" values; nothing tests the fine or the boundary.
- **B** is wrong: all four are typical "fine" values; the boundary is never tested.
- **D** is wrong: running the same value four times gives the same result four times. It tests only one side of the boundary.

If you chose differently, reread *3.2.4 Testing With a Variety of Conditions*.

Question 8

The test with **50** reveals the error, and it is a **boundary case**: 50 is exactly where the rule changes. The rule says \$50 or more ships free, so 50 should print `free shipping`.

Line 11 uses `>`, which is `false` when `orderTotal` is exactly 50. The corrected line is:

```
1 if (orderTotal >= 50)
```

With that change, the same three tests print:

```
20: shipping costs $5
80: free shipping
50: free shipping
```

The original compiled and ran without crashing; it was a logic error, so only a test at the boundary could expose it. The tests at 20 and 80 passed with both versions. Reread *3.2.4 Testing With a Variety of Conditions* if you picked 20 or 80.

Question 9

C. Intellectual property is a creation of the mind, and a program is one. Its owner has the right to decide how it is copied and used.

- **A** is wrong: intellectual property covers creations of the mind, such as songs, stories, and programs, not only physical things.
- **B** is wrong: posting code makes it visible, not free. Permission comes from the owner, not from where the code appears.
- **D** is wrong: renaming variables leaves the other person's work in place. It is still their code.

If you chose differently, reread *3.2.5 Intellectual Property and the Law*.

Question 10

C. Code that is not open source requires the owner's permission, and often a purchase, before it is included.

- **A** is wrong: giving your program away for free does not give you permission to use someone else's code.
- **B** is wrong: disguising copied code does not make it yours or give you permission.
- **D** is wrong: giving credit may be a condition of some open-source licenses, but credit alone does not replace permission for code that is not open source.

If you chose differently, reread *3.2.6 Reusing Code: Open Source and Permission*.

Question 11

Model answer: The rules depend on how the owner published the code, not just on who wrote it. Open-source code was published so anyone may use it for free, as long as they follow its license. Code a company sells is not published that way, so a programmer must get the company's permission, and usually pay, before using it.

A complete answer must:

- say open-source code is free to use (following its license), and
- say other code requires permission and often a purchase.

Reread *3.2.6 Reusing Code: Open Source and Permission* if your answer missed either point.

Question 12

B. Captions help viewers who cannot hear the audio, a social benefit. Speakers with accents are misrepresented and their messages are lost, a harm that falls on a different group. One program, two kinds of impact.

- **A** is wrong: clear English speech is inside the stated conditions, and the app works well there.
- **C** is wrong: a benefit for one group does not cancel a harm for another.

- **D** is wrong: a program that compiles and runs but gives poor results does not have a syntax error. The compiler only checks whether code follows Java's rules.

Reread 3.2.1 *Programs Affect People* and 3.2.3 *System Reliability* if you chose differently.

Question 13

The loop runs with `temperature` equal to 66, 67, 68, and 69:

temperature	temperature < 68	Printed	Expected
66	true	66: heat on	heat on
67	true	67: heat on	heat on
68	false	68: heat off	heat on
69	false	69: heat off	heat off

```
66: heat on
67: heat on
68: heat off
69: heat off
```

It does **not** meet the specification: at the boundary value 68, the heat should be on but is off. The condition should be `temperature <= 68`. Because the tester included the boundary (68) and the value just past it (69), the mistake showed up. Reread 3.2.4 *Testing With a Variety of Conditions* if your table differed.

Question 14

Model answer: The tests all came from one kind of situation, so they say nothing about how the app behaves in others. If the app's stated conditions include the whole growing season, a dry summer week is inside those conditions but was never tested, and the app might give advice that lets crops die. They should add a long dry heat wave (and a boundary case such as exactly the rainfall amount where the advice should switch).

A complete answer must:

- explain that passing tests from only one kind of condition does not show the program works under all its stated conditions, and
- name a specific different condition (dry weather, extreme heat, a boundary rainfall amount, or zero rainfall).

Reread 3.2.4 *Testing With a Variety of Conditions* if your answer missed either point.

Question 15

B. The app was built to save drivers money. Sending everyone to one station, so that its lines block a road, is a harm its designers did not plan, and it lands even on drivers who never used the app.

- **A** is wrong: nothing says the prices were wrong. The app did exactly what it promised; the harm comes from many people following it at once.
- **C** is wrong: the blocked road has nothing to do with copying or reusing someone else's code.
- **D** is wrong: drivers stuck behind the line on the blocked road are worse off, so the effect is not beneficial for all drivers.

Reread 3.2.2 *Unintended Harmful Effects* if you chose differently.

Question 16**Model answer:**

1. **Yes**, it may be used for free, because it is open source. The student must follow the license by keeping the copyright notice.
2. **Not yet**. It is not open source, so the student must get the company's permission and probably pay, even though the game is free.
3. **Not yet**. It is the hobbyist's intellectual property, and nothing gives permission to use it. Being online does not make it free. The student should ask the hobbyist for permission or find other code.

A complete answer must handle all three: open source used by following its license; the sold engine needing permission and likely purchase; the unlabeled code needing permission because visibility is not permission.

Reread 3.2.5 *Intellectual Property and the Law* and 3.2.6 *Reusing Code: Open Source and Permission* if you treated any of them differently.